Langages de scénarios : Utiliser des ordres partiels pour modéliser, vérifier et superviser des systèmes parallèles et répartis.

Thomas Gazagnaire

N° d'ordre: 3708

THÈSE

Présentée devant

devant l'Université de Rennes 1

pour obtenir

le grade de : Docteur de l'Université de Rennes 1
Mention Informatique

par

Thomas Gazagnaire

Équipe d'accueil : DistribCom - IRISA
École Doctorale : Matisse
Composante universitaire : IFSIC

Titre de la thèse :

Langages de scénarios
Utiliser des ordres partiels pour modéliser, vérifier et superviser des systèmes parallèles et répartis.

soutenue le 27 mars 2008 devant la commission d'examen

Philippe	Darondeau	Président
Jean-Michel	Couvreur	Rapporteur
Marc	Zeitoun	Rapporteur
Thierry	Massart	Examinateur
Loïc	Hélouët	Encadrant de Thèse
Claude	Jard	Directeur de Thèse

Nombreuses sont les personnes que je tiens à remercier et qui m'ont accompagnées tout au long de mon doctorat.

Je remercie tout d'abord les membres du jury, pour avoir accepté d'évaluer mon travail : Jean-Michel Couvreur et Marc Zeitoun pour avoir rapporté mon document et Thierry Massart pour avoir participé à mon jury. Je tiens à remercier Philippe Darondeau pour avoir présidé mon jury, pour avoir toujours répondu avec gentillesse et précision à mes nombreuses questions tout au long de mon doctorat et pour avoir relu en détail et commenté ce document. Je remercie aussi Claude Jard et Loïc Hélouët pour m'avoir donné l'opportunité de construire ma thèse : tout en me laissant une grande liberté scientifique, ils ont toujours été là pour m'épauler et me diriger afin que que mes travaux restent cohérents. Merci vraiment pour cela.

Je tiens ensuite à remercier des personnes avec qui j'ai travaillé. Tout d'abord Blaise Genest, qui a eu la patience de partager son savoir en répondant à mes (trop ?) nombreuses questions. Nul doute que sa culture scientifique a eu une une grande influence sur mes travaux. I am also very greatful to P.S. Thiagarajan and Shaofa Yang who welcomed me in Singapore and made me discover the wonderful land of traces. Causal HMSC were formalized during this stay, and as they are the core of my PhD, this period was very important to me. Enfin, je tiens à remercier Rémi Morin qui m'a invité quelques semaines dans la belle ville de Marseille, ce qui a conduit à des discussions enrichissantes sur les HMSC causaux et leurs possibles extensions.

Enfin, merci à tous mes amis et à ma famille pour m'avoir supporté et encouragé pendant ces trois années et demi. Je vous dois beaucoup et je ne serais pas grand chose sans vous tous. Merci !

Table des matières

Introduction

Cette thèse se place dans le cadre de la modélisation et de l'analyse de systèmes *parallèles* et *répartis* :

systèmes parallèles, car composés de nombreuses entités indépendantes qui évoluent de façon autonome, avec des interactions plus ou moins régulières ;

systèmes répartis, car le type d'interaction entre les entités va dépendre, pour une topologie donnée, de la distance qui les sépare.

Plus précisément, lorsque l'on se place dans le cadre de systèmes informatiques, cette problématique prend la forme suivante : les entités sont différents programmes ou processus, qui s'exécutent sur des ordinateurs reliés par l'intermédiaire d'un réseau de télécommunication. Dans ce cadre, le parallélisme et la répartition apparaissent par l'intermédiaire de modes spécifiques de fonctionnement, aussi bien au niveau local qu'au niveau global.

D'une part, le parallélisme s'incarne par deux modes de fonctionnement distincts. Au niveau local, les programmes disposent, chacun à leur tour, d'un court temps d'exécution sur le processeur de la machine, ce qui simule un comportement parallèle. Au niveau global, les différents processeurs des machines sont complètement indépendants et peuvent donc s'exécuter en parallèle.

D'autre part, la répartition s'incarne par l'utilisation de deux modes d'interaction distincts. Au niveau local, chaque machine peut mettre en place des mécanismes de mémoire partagée. Ainsi, deux programmes cohabitant sur une même machine peuvent partager efficacement certaines variables, et les utiliser pour mettre en place des protocoles complexes de synchronisation. Cependant, ce type d'interaction est fortement limité par la performance des bus permettant l'accès à la mémoire et il n'est pas réaliste d'envisager une telle utilisation lorsque les distances de communication deviennent trop grandes. C'est pourquoi, au niveau global, le mode d'interaction retenu est celui de la communication par échange asynchrone de messages entre machines. Un programme peut ainsi envoyer un message et continuer à s'exécuter, sans avoir à attendre une réponse qui peut prendre plusieurs secondes à arriver. Ce type d'interaction utilise les protocoles disponibles sur le réseau de communication sous-jacent. En particulier, nous ferons l'hypothèse que les réseaux de communication sont sans perte et permettent de garantir un minimum d'ordre dans la délivrance des messages[1].

Au final, nous allons nous intéresser, dans ce document, à l'étude de systèmes parallèles et répartis qui vont interagir localement par mémoire partagée et globalement par échange de messages. Nous dirons que ces systèmes sont localement asynchrones et globalement communicants.

[1] via l'utilisation de protocoles tels que TCP/IP qui assure que les messages sont reçus dans l'ordre émis.

Problématiques

Dans le cadre des systèmes localement asynchrones et globalement communicants, nous nous intéressons à deux problématiques : la *modélisation* et l'*analyse* de tels systèmes.

Malheureusement, ces systèmes étant, par nature, très complexes, il est, en général, impossible d'obtenir une modélisation précise en même temps que des outils puissants d'analyse. Il faut donc faire un choix : soit proposer des outils de modélisation très expressifs dont l'analyse automatique va se révéler impossible, soit limiter l'expressivité des modèles proposés afin de pouvoir analyser effectivement certaines de leurs propriétés. Ce dernier choix est celui que nous avons adopté tout au long de cette thèse.

Plus précisément, nous avons décidé de limiter la puissance d'expression des modèles considérés afin de réaliser deux types d'analyse automatique sur les modèles produits.

Le premier type d'analyse est la *vérification*. Cette analyse est effectuée à partir d'un modèle du système à analyser et d'une propriété à vérifier sur toutes les exécutions possibles du système décrit par le modèle. L'intérêt de ce type d'analyse est de pouvoir détecter, avant même son exécution, des problèmes éventuels dans la spécification d'un modèle. Pour que l'analyse puisse être automatisée, il est souvent nécessaire de réduire la précision du modèle considéré. Ainsi, en général, sont vérifiées statiquement des propriétés de sûreté qui indiquent qu'aucune exécution du système modélisé ne conduira ce système dans un état non désiré.

Le second type d'analyse est la *supervision*. Comme pour la vérification, cette analyse est, en général, effectuée à partir d'un modèle du système considéré et d'une propriété à vérifier sur ce système. Cependant, dans le cas de la supervision, une unique exécution est considérée : par exemple celle qui a été observée par des capteurs disposés au sein du système parallèle et réparti considéré. Ce problème est donc plus facile que la vérification (qui considère toutes les exécutions possibles). Par conséquent, la supervision devient possible pour des modèles moins abstraits, ou pour des propriétés plus complexes que celles considérées pour la vérification. A ce titre, les deux types d'analyses se complètent parfaitement : étant donné un modèle d'un système parallèle et réparti, il est possible de le vérifier statiquement vis-à-vis de propriétés simples, ce qui donne une certaine garantie avant l'exécution du système, puis de le superviser dynamiquement vis-à-vis de propriétés plus complexes, ce qui donne une garantie plus forte en cours d'exécution.

Solutions proposées

Pour répondre aux problématiques évoquées précédemment, à savoir la modélisation, la vérification et la supervision de systèmes parallèles et répartis, la solution que nous avons retenue et que nous décrivons dans cette thèse est la suivante :

> *Plutôt que de modéliser séparément chaque entité d'un système parallèle et réparti, puis d'analyser les comportements qui peuvent se produire lorsque ces entités interagissent, nous fournissons une théorie permettant de modéliser et d'analyser globalement le système considéré.*

Nous qualifions cette approche d'"approche globale" ou d'"approche de haut niveau", en opposition à la description indépendante de chaque composant du système, que l'on pourrait qualifier d'"approche locale" ou "approche de bas niveau".

Nous montrons, dans ce document, que l'approche globale permet de résoudre des problèmes d'analyse réputés difficiles pour les systèmes parallèles et répartis. En particulier, la mise au point de ces analyses ne demande pas d'hypothèses trop fortes sur les media d'interaction employés, comme c'est le cas pour les modèles de bas niveau. Par exemple, pour les modèles communicants de bas niveau, il est souvent nécessaire, pour réaliser ces analyses, d'avoir une borne sur la taille des canaux de communication sous-jacents.

La figure 1 décrit schématiquement notre approche et l'organisation du document. Tout d'abord, dans la partie inférieure de cette figure, sont représentés les modèles classiques de bas niveau qui permettent de décrire indépendamment le comportement de chaque entité. Pour ces modèles, il est difficile de répondre aux questions de vérification, représentées sur la droite de cette figure et à celles de supervision, représentées sur sa gauche. Ensuite, la partie supérieure de cette figure représente l'approche différente que nous proposons, qui s'appuie sur une modélisation globale des systèmes considérés. Ces modèles globaux sont, dans certains cas, plus simples à vérifier et à superviser. Enfin, cette figure indique que, d'une part, la partie I de ce document (chapitre 1 et 2), est consacrée à la comparaison du pouvoir d'expression des modèles de haut niveau par rapport à ceux de bas niveau. Elle indique, d'autre part, que la partie II (chapitre 3 et 4), est consacrée à la vérification des modèles globaux. Finalement, elle indique que la partie III (chapitre 5 et 6) s'intéresse à la supervision de ces modèles.

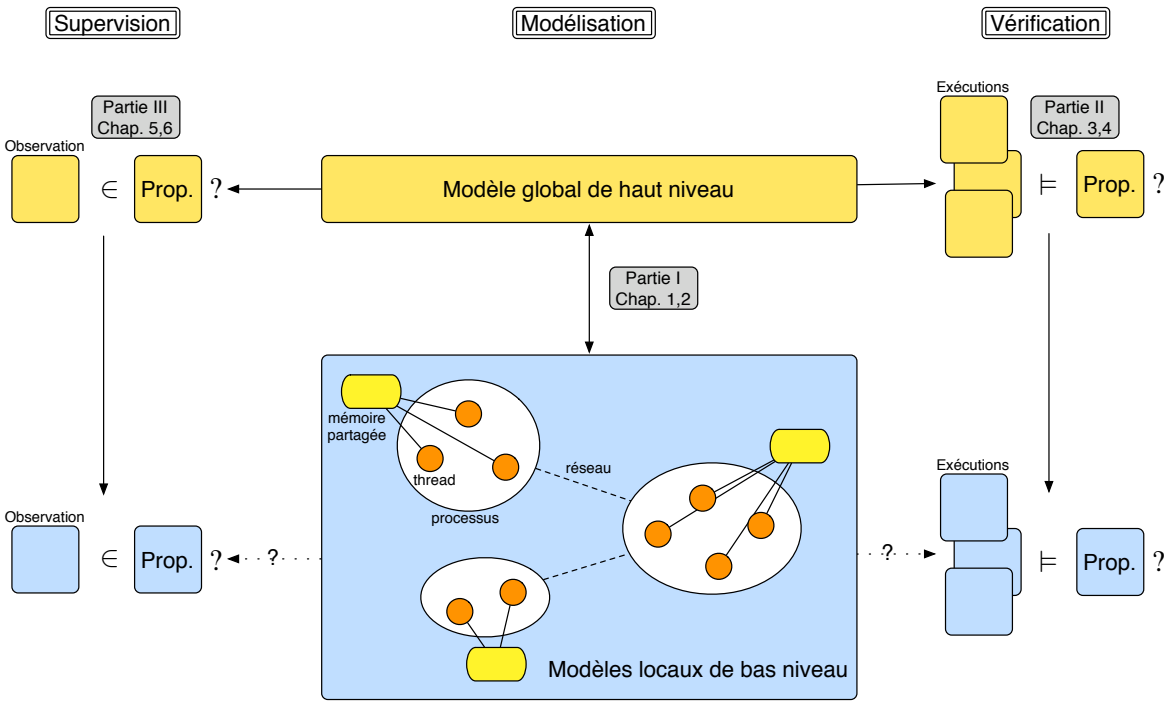


FIG. 1 – *Problématiques de la thèse et organisation du document.*

Utiliser des ordres partiels ?

Les modèles de haut niveau que nous considérons dans cette thèse s'appuient sur la théorie des ordres partiels étiquetés. Ces structures mathématiques sont aussi appelées "modèles d'ordres partiels", car elles permettent de modéliser un système sans en décrire explicitement les entrelacements produits par son exécution séquentielle.

Nous pensons que la théorie des ordres partiels étiquetés est le formalisme idéal à employer dans le cadre des systèmes parallèles et répartis.

En effet, d'une part, ce formalisme permet de modéliser graphiquement, et donc de manière intuitive, des interactions complexes entre différentes entités. D'autre part, cette représentation est très compacte[2], comparée aux modèles de bas niveau qui décrivent des entrelacements d'exécution. Ensuite, ce formalisme permet de s'abstraire des modèles classiques d'exécution séquentielle, ce qui permet d'identifier de nouvelles classes de systèmes où des analyses automatiques vont être possibles. Enfin, il permet de répondre à des questions (liées à la causalité) que les modèles séquentiels classiques peuvent difficilement ou pas du tout traiter.

Organisation du document

Ce document est découpé en trois parties. La première partie concerne la modélisation de haut niveau de systèmes parallèles et répartis, la seconde la vérification de tels modèles, et la troisième leur supervision. Plus précisément, ce document s'organise comme suit :

Première partie (chapitres 1 et 2) : Modélisation

Chapitre 1 : Panorama des modèles de parallélisme

Dans ce chapitre, nous donnons les résultats déjà existants sur la modélisation de systèmes. Nous partons des modèles associés aux systèmes séquentiels classiques, pour ensuite présenter les modèles associés aux systèmes asynchrones, qui interagissent par mémoire partagée. Nous présentons finalement les structures d'ordre partiel qui permettent de modéliser les systèmes parallèles qui communiquent par échange de messages asynchrones. Nous préciserons, pour chacun de ces modèles, les analyses automatiques qu'il est possible de réaliser.

Chapitre 2 : Modèles mixtes

Dans ce chapitre, nous présentons les premiers résultats de cette thèse. Nous commençons par introduire une nouvelle famille de langages de modélisation, appelée HMSC causaux, qui étend le formalisme des HMSC (pour "High-Level Message Sequence Charts", un langage de spécification normalisé pour les systèmes communicants) et les traces de Mazurkiewicz (un formalisme qui permet de décrire des systèmes asynchrones). Ce formalisme permet de décrire une large classe de systèmes parallèles et répartis. Nous montrons ensuite des résultats de traduction entre des sous-classes syntaxiques des HMSC causaux vers des modèles de bas

[2]La taille gagnée est de l'ordre d'un facteur exponentiel.

niveau séquentiels, vers des modèles de bas niveau communicants et vers des modèles de bas niveau dits mixtes, car localement asynchrones et globalement communicants. Les résultats énoncés dans ce chapitre, sont, à notre sens préliminaires : ils permettent uniquement de caractériser des systèmes parallèles dont les canaux de communication ont des capacités bornées, ce qui n'est pas totalement satisfaisant. A long terme, obtenir une caractérisation de systèmes non bornés, s'appuyant par exemple sur le modèle opérationnel mixte proposé dans ce chapitre, est un enjeu majeur afin de caractériser l'expressivité exacte des modèles que nous proposons dans ce document.

Deuxième partie (chapitre 3 et 4) : Vérification

Chapitre 3 : Vérification de modèles

Ce chapitre présente les principaux résultats théoriques que nous avons obtenus sur la vérification de modèles de haut niveau, décrits à l'aide de générateurs de familles d'ordres partiels étiquetés. Nous présentons dans ce chapitre deux résultats principaux.

Tout d'abord, le premier résultat s'appuie sur l'extension aux ordres partiels des logiques temporelles classiques. Nous montrons, d'une part, qu'il est possible de caractériser les familles de modèles analysables par ces logiques, et d'autre part, que la complexité de ce genre d'analyse reste équivalente à celle du model-checking de modèles séquentiels. C'est un résultat très satisfaisant, car il indique que la concision des modèles d'ordres partiels est réelle et peut être conservée tout au long de l'analyse automatique.

Ensuite, le second résultat est que, si nous décrivons les propriétés à analyser dans une restriction du langage de modélisation, il est possible d'analyser n'importe quel modèle d'ordres partiels, cette fois sans aucune restriction syntaxique. Ce résultat montre l'intérêt d'utiliser ce type de modèles. Enfin, nous utilisons ces résultats pour caractériser les analyses de vérification décidables pour les HMSC causaux.

Chapitre 4 : Vérification partielle de modèles

Dans ce chapitre, nous définissons un nouveau modèle, appelé "boxed pomsets", qui nous sert, ensuite, à faire de la vérification partielle de modèles (seulement dans le cas de la vérification de propriétés positives).

D'une part, nous montrons que ce modèle possède de bonnes propriétés de projection, puisque la projection de la composition de deux boxed pomsets est la composition de la projection de ces deux boxed pomsets : les rationnels des boxed pomsets sont donc stables par projection.

D'autre part, nous traduisons le problème de la vérification positive et partielle de modèles, donnés sous la forme de rationnels de pomsets, en un nouveau problème. Celui-ci consiste à décider si le langage engendré par une expression rationnelle de boxed pomsets est équivalent à celui d'une expression rationnelle de pomsets. Nous montrons que ce nouveau problème est décidable pour les boxed pomsets et donc, que la vérification positive et partielle de modèles utilisant des pomsets est décidable.

Troisième partie (chapitre 5 et 6) : Supervision.

Chapitre 5 : Supervision avec observation complète

Ce chapitre se focalise sur le traitement efficace de fichiers, dont la taille peut être très importante, issus de l'observation exhaustive des événements d'un système parallèle et réparti. Pour cela nous avons développé deux techniques.

La première technique consiste à compresser une exécution en même temps que celle-ci est observée. La compression est paramétrée par une fonction de typage et le résultat de cette compression est une abstraction de l'exécution initiale, dans laquelle les événements de même type sont regroupés dans des macro-événements, de telle sorte que l'abstraction reste un ordre partiel. Cette technique est un pré-traitement intéressant et transparent pour la plupart des méthodes que nous développons.

La seconde technique consiste à compresser un ordre partiel étiqueté en oubliant l'ordre qu'il définit et en comptant le nombre de toutes les étiquettes qu'il contient. Ensuite, il est possible de caractériser les classes de modèles pour lesquels on sait efficacement si une exécution ayant exactement le même décompte existe. Nous résolvons ce problème en utilisant un encodage adéquat vers des problèmes matriciels.

Chapitre 6 : Supervision avec observation partielle

Finalement, le dernier chapitre de ce document est consacré à la supervision de systèmes que l'on n'est pas capable d'observer entièrement. Plus précisément, étant donné un modèle d'un système et une observation partielle d'une exécution du système qu'il spécifie, nous cherchons à inférer les différents éléments qui n'ont pas été observés. Ces éléments sont de deux natures : des événements et des causalités.

La première technique que nous développons, appelée diagnostic, consiste à inférer de l'observation partielle et d'un modèle du système observé, toutes les exécutions complètes dont la projection sur les événements observables expliquent l'observation. Comme il existe, a priori, un ensemble infini de ces exécutions, nous montrons qu'il est possible de construire un générateur fini qui engendre exactement les exécutions du modèle dont la projection explique l'observation.

La seconde technique que nous développons, appelée corrélation d'événements, consiste à inférer de l'observation partielle et d'un modèle du système observé, toutes les causalités dont nous sommes sûrs qu'elles se sont produites. Le résultat de cette technique est une extension de l'observation, extension dans laquelle nous avons rajouté toutes les causalités inférées par le modèle. Ceci permet de réduire le nombre d'éléments minimaux de l'observation et donc de diminuer le nombre de causes possibles du phénomène observé.

Première partie

Modélisation

Chapitre 1

Panorama des modèles parallèles

Ce chapitre constitue, avec le chapitre suivant, la partie I de cette thèse, consacrée aux *modèles de parallélisme*.

Nous présentons, dans ce chapitre, différents modèles qui permettent de spécifier des systèmes informatiques complexes. Nous nous plaçons pour cela dans le cadre mathématique des monoïdes, structures qui vont nous servir de cadre pour modéliser le comportement de haut niveau de ces systèmes. Dans ce cadre, nous précisons deux types de modélisation pour les systèmes parallèles et répartis, déjà largement étudiés ces dernières années.

Le premier type de modélisation que nous décrivons s'attache à spécifier des systèmes asynchrones, systèmes composés d'entités qui interagissent par mémoire partagée. Le monoïde correspondant à ces systèmes est celui du monoïde de traces. Les modèles de haut niveau correspondants sont, donc, les expressions rationnelles du monoïde de traces, les modèles de bas niveau étant les réseaux d'automates asynchrones.

Le second type de modélisation que nous décrivons, s'attache à spécifier des systèmes communicants, systèmes composés d'entités qui interagissent par échange de messages. Les monoïdes correspondants sont les monoïdes des HMSC ou celui des CHMSC. Les modèles de haut niveau sont, là aussi, les expressions rationnelles de ces monoïdes et les modèles de bas niveau sont les réseaux d'automates communicants.

Dans le chapitre suivant, lui aussi consacré aux modèles, nous présenterons les premiers résultats de cette thèse en donnant des modèles qui généralisent les deux approches précédentes et permettent de spécifier des interactions mixtes.

Enfin, les parties II et III de ce document seront consacrées, respectivement, à la vérification et à la supervision de ces systèmes parallèles et répartis mixtes.

Contexte

Dans le cadre des systèmes séquentiels à événements discrets, le lien entre les langages de spécification et les modèles opérationnels est bien compris. En particulier, [Kleene 56] a montré que ces deux formalismes permettaient de décrire les mêmes structures. Un concepteur dispose donc, d'un côté, d'outils simples et expressifs permettant de décrire le fonctionnement d'un système et de l'autre, de modèles d'exécution formellement définis et pour lesquels de nombreux problèmes de vérification sont décidables statiquement.

Cependant, dans le cadre des systèmes parallèles et répartis à événements discrets, le tableau est plus complexe. En effet, la plupart des modèles qui ont été proposés ne sont pas assez expressifs car ils ne permettent pas d'exprimer certains aspects de la répartition, ou inutilisables en pratique car de nombreux problèmes (de vérification) sont indécidables ou très peu efficaces, pour ces modèles. Trouver des modèles qui ne tombent dans aucun de ces travers est l'enjeu principal de cette thèse. Mais, avant de présenter nos résultats, nous rappelons, dans ce chapitre, la définition des principaux modèles de haut niveau que nous étendons dans cette thèse, et nous donnons les liens qui existent avec certains modèles de bas niveau plus classiques.

Dans ce chapitre, nous présentons donc plusieurs modèles de spécification et d'exécution, en particulier ceux liés aux systèmes séquentiels et leurs extensions aux systèmes parallèles interagissant par mémoire partagée ou par échange de messages asynchrones.

Nous commençons par donner le cadre général de modélisation dans lequel nous nous plaçons et par fixer les définitions qui vont être utilisées tout au long de cette thèse.

Un cadre général

Les monoïdes (ou semi-groupes avec un élément neutre) sont des structures algébriques dont les éléments ne sont pas inversibles. Ces structures ont été étudiées par la communauté naissante de l'informatique théorique de l'après-guerre, par exemple dans [Schützenberger 55] où les premières notions concernant les semi-groupes sont étudiées, ou encore dans [Clifford 61] qui est le premier ouvrage de synthèse concernant ces structures. En effet, les monoïdes permettent de modéliser le comportement de systèmes qui évoluent au cours du temps, comme c'est le cas pour les systèmes informatiques en général.

Plus précisément, le lien entre les monoïdes et les systèmes informatiques est le suivant : ces systèmes ne sont pas réversibles et l'évolution du temps se traduit par une perte d'informations. Ceci a pour conséquence que les actions produites par de tels systèmes ne peuvent pas être annulées. L'absence d'inverse dans les monoïdes répond donc parfaitement à ce besoin de modélisation.

Plus formellement, un *monoïde* $(\mathbb{M}, \cdot, 1)$, noté plus simplement $\mathbb{M}$ est une structure algébrique composée d'un ensemble $\mathbb{M}$ qui est muni d'une loi de composition interne associative $\cdot$ et d'un élément neutre 1. En d'autres termes :

Définition 1.1 (monoïde) *Une structure* $\mathbb{M} = (\mathbb{M}, \cdot, 1)$ *est un monoïde si elle vérifie les trois axiomes suivants :*

(loi de composition interne) $\forall x, y \in \mathbb{M},\ x \cdot y \in \mathbb{M}$;

(élément neutre) $\forall x \in \mathbb{M},\ x \cdot 1 = 1 \cdot x = x$;

(associativité) $\forall x, y, z \in \mathbb{M},\ (x \cdot y) \cdot z = x \cdot (y \cdot z)$.

Etant donné deux monoïdes $\mathbb{M}$ et $\mathbb{M}'$, une fonction $\eta : \mathbb{M} \to \mathbb{M}'$ est un *morphisme* du monoïde $\mathbb{M}$ vers le monoïde $\mathbb{M}'$ si, pour tous éléments $x, y \in \mathbb{M}$, $\eta(1) = 1$ et $\eta(x \cdot y) = \eta(x) \cdot \eta(y)$. Un *isomorphisme* est un morphisme bijectif.

Etant donné un monoïde $\mathbb{M}$, une *congruence* de $\mathbb{M}$ est une relation d'équivalence $\sim \subseteq \mathbb{M} \times \mathbb{M}$ compatible avec la loi de composition interne. Plus précisément, une congruence vérifie les trois axiomes d'une relation d'équivalence, ainsi qu'un axiome de compatibilité de la loi de composition. Plus formellement, $\sim$ est une congruence si elle vérifie :
(réflexivité) $\forall x \in \mathbb{M},\ x \sim x$;
(symétrie) $\forall x, y \in \mathbb{M},\ x \sim y \Leftrightarrow y \sim x$;
(transitivité) $\forall x, y, z \in \mathbb{M},\ (x \sim y) \wedge (y \sim z) \Rightarrow x \sim z$;
(compatibilité) $\forall x, y, x', y' \in \mathbb{M},\ x \sim x' \wedge y \sim y' \Rightarrow x \cdot y \sim x' \cdot y'$.

Etant donné un monoïde $\mathbb{M}$ et une relation d'équivalence $\sim$ de $\mathbb{M}$, la *classe d'équivalence* d'un élément $x \in \mathbb{M}$, notée $[x]_\sim$, est l'ensemble des éléments de $\mathbb{M}$ équivalents à x par la relation $\sim$, c'est-à-dire $[x]_\sim = \{y \in \mathbb{M} \mid x \sim y\}$. De même, étant donné $L \subseteq \mathbb{M}$, le quotient de L par la relation d'équivalence $\sim$ est notée $[L]_\sim = \{[x]_\sim \mid x \in L\}$.
Une congruence $\sim$ de $\mathbb{M}$ induit naturellement un monoïde quotient $(\mathbb{M}/_\sim, \cdot, [1]_\sim)$, avec $\mathbb{M}/_\sim = \{[x]_\sim \mid x \in \mathbb{M}\}$ et $[x]_\sim \cdot [y]_\sim = [x \cdot y]_\sim$. Pour des raisons de lisibilité, un tel monoïde quotient est simplement noté $\mathbb{M}/_\sim$.

Etant donné une relation R, la fermeture transitive de R, notée R^*, est la plus petite relation qui vérifie : si x, y et z sont tels que $x\ R\ y$ et $y\ R\ z$, alors $x\ R^*\ z$. La réduction transitive de R est une relation minimale S, vérifiant $S^* = R$.

Pour tous les sous-ensembles L et L' d'un monoïde $\mathbb{M}$, le produit de L et L' est $L \cdot L' = \{x \cdot x' \mid x \in L \wedge x' \in L'\}$. Nous notons $L^0 = \{1\}$, et pour tout entier positif n, $L^{n+1} = L^n \cdot L$. L'itération de L est notée $L^\star = \bigcup_{n \in \mathbb{N}} L^n$, qui est un sous-monoïde de $\mathbb{M}$. Un ensemble L est *finiment engendré* dans $\mathbb{M}$ par un ensemble fini $X \subseteq \mathbb{M}$, ce qui est noté $L \subseteq \langle X \rangle_\mathbb{M}$, si il existe un sous-ensemble fini $X \subseteq \mathbb{M}$ tel que $L \subseteq X^\star$. Un monoïde $\mathbb{M}$ est dit finiment engendré si $L = \{x \in \mathbb{M}\}$ est finiment engendré dans $\mathbb{M}$.

Définition 1.2 (rationnels) *Soit* $\mathbb{M}$*, un monoïde. L'ensemble* $REX(\mathbb{M})$ *des* expressions rationnelles *de* $\mathbb{M}$ *est donné par la syntaxe suivante :* $\alpha ::= \emptyset \mid m \mid \alpha_1 \cdot \alpha_2 \mid \alpha_1 + \alpha_2 \mid \alpha^\star$*, où* $m \in \mathbb{M}$*. Le* langage *d'une expression rationnelle, noté* $\mathcal{L}_\mathbb{M} : REX(\mathbb{M}) \to 2^\mathbb{M}$*, est défini inductivement de la manière suivante :*

- $\mathcal{L}_\mathbb{M}(\emptyset) = \emptyset$;
- $\mathcal{L}_\mathbb{M}(m) = \{m\}$;
- $\mathcal{L}_\mathbb{M}(\alpha_1 \cdot \alpha_1) = \mathcal{L}_\mathbb{M}(\alpha_1) \cdot \mathcal{L}_\mathbb{M}(\alpha_2)$;

- $\mathcal{L}_{\mathbb{M}}(\alpha_1 + \alpha_1) = \mathcal{L}_{\mathbb{M}}(\alpha_1) \cup \mathcal{L}_{\mathbb{M}}(\alpha_2)$;
- $\mathcal{L}_{\mathbb{M}}(\alpha^\star) = \mathcal{L}_{\mathbb{M}}(\alpha)^\star$.

Un sous-ensemble $L \subseteq \mathbb{M}$ est rationnel *si il est le langage d'une expression rationnelle. De manière équivalente, L est rationnel si il peut être obtenu à partir d'une suite finie d'unions, de produits, et d'itérations de sous-ensembles de $\mathbb{M}$. Les rationnels de $\mathbb{M}$ sont notés $RAT(\mathbb{M})$.*

Par la suite, nous écrirons $\alpha_1\alpha_2$ à la place de $\alpha_1 \cdot \alpha_2$, et l'opérateur $\star$ est prioritaire sur $\cdot$, lui même prioritaire sur $+$. Remarquons que, pour tout morphisme $\eta : \mathbb{M} \to \mathbb{M}'$, si $L \subseteq \mathbb{M}$ est rationnel, alors $\eta(L) = \{\eta(l) \mid l \in L\} \subseteq \mathbb{M}'$ est rationnel.

La proposition suivante donne l'équivalence de pouvoir expressif entre les différents modèles présentés jusqu'à présent. Plus précisément, elle indique que les langages d' expressions rationnelles, les sous-ensembles rationnels et les sous-ensembles finiment engendrés d'un monoïde quelconque $\mathbb{M}$ coïncident.

Proposition 1.3 ([Diekert 95]) *Soient $\mathbb{M}$, un monoïde et $L \subseteq \mathbb{M}$. Alors, les trois affirmations suivantes sont équivalentes :*

- *L est rationnel, c'est-à-dire $L \in RAT(\mathbb{M})$;*
- *L est le langage d'une expression rationnelle de $REX(\mathbb{M})$;*
- *L est finiment engendré, c'est-à-dire il existe $X \subseteq \mathbb{M}$ fini, tel que $L \subseteq \langle X \rangle_{\mathbb{M}}$.*

Ces différentes formulations nous donnent des outils expressifs et abstraits pour définir des comportements infinis, à l'aide d'opérateurs d'itération et de choix, dans le cadre des monoïdes. Ainsi, nous considérons, dans tout ce document, l'hypothèse suivante :

Un modèle de haut niveau est une expression rationnelle d'un monoïde bien choisi.

Il faut donc définir les monoïdes adaptés à la modélisation de haut niveau, ainsi que les modèles opérationnels (ou de bas niveau) induits par les hypothèses faites sur le système sous-jacent. Enfin, il reste à faire le lien entre ces deux types de représentation et à caractériser les analyses décidables pour ces modèles.

Organisation du chapitre

Dans la suite de ce chapitre, nous répondons rapidement à ces questions pour le cadre séquentiel dans la partie 1.1, où nous introduisons le monoïde libre et où les modèles d'exécution sont les automates. Ensuite, dans la partie 1.2, nous nous plaçons dans le cadre des systèmes parallèles interagissant par mémoire partagée. Dans ce cadre, le monoïde considéré est le monoïde de traces et les modèles de bas niveau sont les réseaux d'automates asynchrones. Enfin, dans la partie 1.3, nous étudions les systèmes parallèles interagissant par échange de messages (asynchrones). Le monoïde correspondant est le monoïde des ordres partiels étiquetés décrivant l'interaction entre différents processus s'échangeant des messages et les modèles opérationnels y sont appelés des automates communicants.

1.1 Modèles séquentiels

Nous présentons, maintenant, une spécialisation des définitions présentées dans la partie précédente à des modèles permettant de décrire l'évolution de systèmes à événements discrets où le contrôle de l'exécution est global, appelés aussi systèmes séquentiels. De tels systèmes produisent des séquences d'actions, et le choix de la future action à exécuter est effectué par un unique "intervenant" qui centralise le contrôle du flot d'exécution du système. De tels systèmes permettent, par exemple, de décrire le fonctionnement d'un ordinateur où toutes les instructions sont effectuées par une unique entité, le processeur. Le cadre des systèmes séquentiels a été largement étudié, depuis une cinquantaine d'années, et de nombreux résultats existent. Dans cette partie, nous donnons, tout d'abord, des définitions générales concernant les langages de spécification, à l'aide d'expressions rationnelles du monoïde libre (partie 1.1.1). Ensuite, dans la partie 1.1.2, nous définissons les automates dans le cadre des monoïdes finiment engendrés, qui correspondent aux modèles opérationnels des langages séquentiels. Enfin, nous montrons, dans la partie 1.1.3, que les langages de spécification et les modèles d'exécution ont le même pouvoir d'expression pour les modèles séquentiels, et nous présentons des résultats de décidabilité sur ces modèles. Ces résultats motivent leur utilisation pour modéliser et analyser des systèmes séquentiels.

1.1.1 Monoïde libre

Dans la suite de ce document, nous utilisons une classe très particulière de monoïdes. Cette classe se place dans le cadre discret, lorsque chaque action effectuée par le système est décomposable en actions atomiques. Nous nous intéressons donc à une classe de monoïdes finiment engendrés. L'ensemble des actions que le système à modéliser peut effectuer est regroupé dans un ensemble Σ, appelé *alphabet*. Les éléments $a, b, \ldots$ de Σ sont donc appelés indifféremment des *actions* ou des *lettres* et les séquences finies de lettres $a_1 \ldots a_n$, des *mots*. La séquence vide est notée ε. Enfin, l'ensemble des mots composés à partir d'un alphabet Σ est noté $\Sigma^\star$. Pour tout $u \in \Sigma^\star$, nous notons $\Sigma(u)$ l'ensemble des lettres qui apparaissent dans u.

De plus, la classe de monoïdes à laquelle nous nous intéressons modélise des systèmes à mémoire, où chaque action effectuée est enregistrée et fait partie de la production du modèle : la loi de composition doit refléter ce fait. Ainsi, afin d'interpréter $\Sigma^\star$ comme un monoïde, nous lui associons la relation de *concaténation*, notée $\cdot$, et définie de la manière suivante : $a_1 \ldots a_n \cdot b_1 \ldots b_m = a_1 \ldots a_n b_1 \ldots b_m$, pour tout mot $a_1 \ldots a_n$ et $b_1 \ldots b_m$. Cette relation est une loi interne, associative et qui admet ε comme élément neutre : en conséquence $(\Sigma^\star, \cdot, \varepsilon)$ est un monoïde, appelé *monoïde libre*.

Définition 1.4 (monoïde libre) *Soient Σ, un ensemble d'actions, ε, le mot vide et $\cdot$, défini tel que $u \cdot v = uv$ pour tout mot u, v dans $\Sigma^\star$. Alors, $\Sigma^\star = (\Sigma^\star, \cdot, \varepsilon)$ est le* monoïde libre. *Les sous-ensembles de $\Sigma^\star$ sont appelés des* langages.

Pour conclure, étant donné un monoïde $\mathbb{M}$ explicitant toutes les actions possibles pour un système, une expression rationnelle de $\mathbb{M}$ peut être vue comme la spécification d'un modèle décrivant l'évolution, au cours du temps, du système que l'on cherche à modéliser. Dans la partie suivante, nous présentons un modèle d'exécution pour ce type de spécifications.

1.1.2 Automates

La théorie des automates est née quelques années avant celle des monoïdes présentée dans le chapitre précédent, et le lien entre ces deux représentations a été fait par [Schützenberger 55]. Lorsque Shanon publie [Shannon 48], il fonde non seulement la théorie de l'information, mais sa notion de "discrete noiseless system" utilisée pour décrire les canaux de communication, combinée avec une chaîne de Markov, est, en fait, exactement la définition moderne d'un automate. Puis, Von Neuman introduit le terme "théorie des automates" dans une conférence ([vonNeumann 51]) où il expose les perspectives d'une future théorie. Enfin, citons les travaux fondateurs de Kleene dans [Kleene 56], qui reprend et étend un article de McCulloch et Pitts ([McCulloch 43]) sur les réseaux de neurones. Pour un historique plus complet, se reporter, par exemple, à [Perrin 95].

Formellement, un automate peut se définir de la manière suivante :

Définition 1.5 ($\mathbb{M}$-automate) *Etant donné un monoïde libre $\mathbb{M}$, un* $\mathbb{M}$-automate $\mathcal{A}$ *est une structure $(S, \rightarrow, M, S_i, S_f)$ où :*
- *S est un ensemble d'*états *;*
- *$\rightarrow$: $S \times M \times S$ est une relation de* transition *;*
- *$M \subseteq \mathbb{M}$ est un ensemble d'*étiquettes *;*
- *$S_i \subseteq S$ est un ensemble d'*états initiaux *;*
- *$S_f \subseteq S$ est un ensemble d'*états finaux.

Dans le cas où S_i est un singleton et $\rightarrow$ une fonction de $S \times M$ dans S, $\mathcal{A}$ est appelé un automate *déterministe*.

Un *chemin* ρ de $\mathcal{A}$ est une succession de transitions de $\rightarrow$ consécutives, c'est-à-dire telles que $\rho = n_0 \xrightarrow{l_1} n_1 \ldots \xrightarrow{l_k} n_k$ et (n_i, l_{i+1}, n_{i+1}) est dans $\rightarrow$. Est associé, à chaque chemin ρ, l'élément $\mathcal{L}_{\mathbb{M}}(\rho) = l_1 \cdot \ldots \cdot l_k$ de $\mathbb{M}$. Un *chemin acceptant* est un chemin débutant dans un état initial et terminant dans un état final. Le langage d'un automate est l'ensemble des éléments de $\mathbb{M}$ définis par des chemins acceptants de A, c'est-à-dire $\mathcal{L}_{\mathbb{M}}(\mathcal{A}) = \{\mathcal{L}_{\mathbb{M}}(\rho) \mid \rho \text{ est un chemin acceptant de } \mathcal{A}\}$. On dira que $\mathcal{A}$ reconnaît L si L est le langage de $\mathcal{A}$.

Grâce à cette définition, nous pouvons introduire, maintenant, la notion de *reconnaissabilité*.

Définition 1.6 (reconnaissabilité) *Soient $\mathbb{M}$, un monoïde et $L \subseteq \mathbb{M}$. L est dit* reconnaissable *dans $\mathbb{M}$ si il existe un monoïde fini $\mathbb{F}$ et un morphisme $\eta : \mathbb{M} \rightarrow \mathbb{F}$ tels que $L = \eta^{-1}(F)$, pour $F \subseteq \mathbb{F}$. Les reconnaissables de $\mathbb{M}$ sont notés $REC(\mathbb{M})$.*

Si $\mathbb{M}$ est le monoïde libre, alors L est reconnaissable si il existe un $\mathbb{M}$-automate $\mathcal{A}$ tel que, $\mathcal{L}_{\mathbb{M}}(\mathcal{A}) = L$. Dans ce cas, L est dit régulier.

Ainsi, dans le cas du monoïde libre, les langages réguliers correspondent aux ensembles engendrés par une machine séquentielle à mémoire finie, qui fonctionne à pas discrets. En effet, il est facile de voir que tout $\Sigma^\star$-automate (dont les transitions sont étiquetées par des mots quelconques de $\Sigma^\star$) est équivalent à un $\Sigma^\star$-automate déterministe étiqueté par des lettres de $\Sigma \cup \{\varepsilon\}$. Chaque tir de transition de ce nouvel automate correspond à l'exécution d'une seule

et unique action atomique du système, et l'état dans lequel est cet automate est entièrement déterminé par la séquence d'actions qu'il vient d'effectuer.

L'intérêt de cette approche est que le point de vue peut être inversé. En effet, nous pouvons voir $\mathcal{A}$, non plus comme un "reconnaisseur", mais comme un "générateur" d'une classe de comportements définis par $\mathcal{A}$. D'où l'intuition que les automates correspondent à un modèle opérationnel d'un système : dans un état donné de $\mathcal{A}$, tirer une transition de la machine génère des modifications admissibles de l'état du système car elles ne font pas sortir le système de la classe de comportements définis.

En résumé, les automates correspondent à un modèle d'exécution servant à décrire le comportement d'un système séquentiel. Nous montrons, dans la partie suivante, le lien qui existe entre langages rationnels de spécification et modèles d'exécution donnés sous la forme d'automates.

1.1.3 Résultats

Nous donnons une série de résultats concernant le monoïde libre, car celui-ci possède de bonnes propriétés structurelles et algorithmiques (alors que ce n'est pas vraiment le cas pour des monoïdes quelconques).

Tout d'abord, le lien entre les ensembles rationnels et reconnaissables d'un monoïde est très fort. En particulier, ces ensembles sont identiques lorsque l'on se limite au monoïde libre :

Théorème 1.7 ([Kleene 56]) *Soit $\Sigma^\star$, un monoïde libre. Alors, $L \subseteq \Sigma^\star$ est régulier si, et seulement si, L est rationnel.*

Kleene a donc montré que, sur ces structures, toute spécification rationnelle est équivalente à un modèle d'exécution, exprimé à l'aide d'un automate fini (et inversement). De plus, la traduction d'un représentation vers une autre peut se faire en taille linéaire par rapport à la taille des entrées. C'est ce qui fait tout l'intérêt du monoïde libre et qui rend ce modèle si utilisé et étudié. En particulier, ceci veut dire que, dans le cadre où le système étudié possède des actions atomiques (c'est-à-dire est "à événements discrets") et où la sortie du système consiste en une trace de l'exécution des différentes actions, tout modèle d'exécution correspondant à un automate peut être décrit par une spécification rationnelle.

Les $\Sigma^\star$-automates sont donc des modèles simples et puissants, qui constituent, avec les expressions rationnelles, la base de la théorie des systèmes séquentiels et discrets. D'un côté, les expressions rationnelles peuvent se comprendre comme un langage de spécification de comportement, et de l'autre les automates comme un modèle opérationnel qui permet d'exécuter la spécification d'un système.

De plus, les ensembles réguliers possèdent de très bonnes propriétés algorithmiques que nous utilisons tout au long de nos travaux. Plus précisément, nous nous intéressons à trois questions théoriques sur ces modèles, qui ont un impact important sur les techniques que nous décrirons par la suite :

Appartenance : Soient $x \in \mathbb{M}$ et α, un modèle. Est-ce que $x \in \mathcal{L}_{\mathbb{M}}(\alpha)$?

Vacuité de l'intersection : Soient α et β, deux modèles. Est-ce que $\mathcal{L}_{\mathbb{M}}(\alpha) \cap \mathcal{L}_{\mathbb{M}}(\beta) = \emptyset$?

Inclusion : Soient α et β, deux modèles. Est-ce que $\mathcal{L}_{\mathbb{M}}(\alpha) \subseteq \mathcal{L}_{\mathbb{M}}(\beta)$?

Proposition 1.8 ([Rozenberg 97]) *Pour les systèmes séquentiels, nous avons les résultats de complexité suivants, triés verticalement par problème et horizontalement par générateur des langages L et L' :*

	$REX(\Sigma^\star)$	$\Sigma^\star$-*automate*
$x \in L$	*NLOGSPACE-complet*	*NLOGSPACE-complet*
$L \cap L' = \emptyset$	*PSPACE-complet*	*PSPACE-complet*
$L \subseteq L'$	*PSPACE-complet*	*PSPACE-complet*

Dans cette proposition, NLOGSPACE est la classe des problèmes résolubles par des machines de Turing non déterministes, dont la mémoire est bornée par une fonction logarithmique de la taille des entrées. PSPACE est la classe des problèmes résolubles par des machines de Turing déterministes, dont la mémoire est bornée par une fonction polynomiale de la taille des entrées. Le lecteur intéressé par plus de précisions concernant la théorie de la complexité pourra se rapporter, par exemple, à [Papadimitriou 94]. De plus, les preuves liées à ces résultats de complexité sont constructives : il existe donc des algorithmes effectifs permettant de savoir si un mot appartient au langage d'un automate, de savoir si deux automates ont le même langage, etc.

1.2 Modèles asynchrones

Dans la partie précédente, nous avons donné les définitions et principaux résultats exprimant le lien fort qui existe entre spécifications et modèles d'exécution pour les systèmes séquentiels. Dans le cas des systèmes asynchrones, c'est-à-dire dans le cas où plusieurs systèmes séquentiels à événements discrets fonctionnent de manière indépendante et interagissent par mémoire partagée et par primitives de synchronisation, les choses deviennent un peu plus complexes. Nous présentons dans cette partie les définitions et résultats principaux concernant ces systèmes.

Plus précisément, nous présentons, dans la partie 1.2.1, le monoïde de traces, dont les expressions rationnelles servent de langage de spécification pour les modèles communiquant par mémoire partagée. Puis nous donnons, dans la partie 1.2.2, les motivations nécessaires pour introduire un nouveau modèle opérationnel, les automates asynchrones. Nous rappelons, enfin, dans la partie 1.2.3, le lien fort qui existe entre une sous-classe syntaxique de spécifications rationnelles et ce nouveau modèle opérationnel : ces deux modèles ont un pouvoir d'expression identique.

1.2.1 Monoïde de traces

L'étude d'une théorie décrivant le fonctionnement de systèmes parallèles a été motivée par les travaux fondateurs de Petri. En effet, dans [Petri 62], celui-ci présente un modèle fondé sur la communication de systèmes séquentiels interconnectés, et il expose de nouveaux problèmes qui apparaissent lorsque l'on considère le parallélisme intrinsèque de tels systèmes. A la fin des années 1960, Cartier et Foata donnent les bases de la théorie des monoïdes partiellement commutatifs dans [Cartier 69]. Quelques années plus tard, motivé par la résolution d'un certain nombre de problèmes posés par les approches fondées sur des modèles entrelacés

de comportements pour décrire des systèmes parallèles, Mazurkiewicz fait la connexion entre les modèles parallèles apportés par Petri et les structures mathématiques étudiées par Cartier et Foata, en développant, dans [Mazurkiewicz 77], la théorie des traces. Cette théorie est devenue une théorie centrale pour modéliser des systèmes parallèles qui communiquent par mémoire partagée. Depuis, cette théorie a été largement explorée (se reporter à [Diekert 95], le livre qui sert de référence dans ce domaine, pour des détails plus précis). Dans cette partie, nous donnons les définitions et théorèmes essentiels dont nous nous servons par la suite.

Soit Σ, un alphabet. Une *relation de dépendance* $D \subseteq \Sigma^2$ est une relation symétrique et réflexive. La relation $I = \Sigma^2 - D$, non réflexive et symétrique, est appelée *relation d'indépendance*. A partir d'un alphabet Σ et d'une relation de dépendance D, nous pouvons définir un couple (Σ, D), appelé *alphabet "concurrent"*. L'alphabet concurrent (Σ, D) induit une relation d'équivalence $\sim_I$ sur les éléments de $\Sigma^\star$: deux mots u et v sont équivalents, ce qui est noté $u \sim_I v$, si l'on peut passer de l'un à l'autre en commutant successivement des lettres voisines indépendantes. Plus formellement, $\sim_I$ est la plus petite congruence de $\Sigma^\star$ telle que $ab \sim_I ba$ pour toute paire $(a, b) \in I$. Ces définitions nous permettent d'introduire la définition du monoïde de traces.

Définition 1.9 (monoïde de traces) *Soient (Σ, D), un alphabet concurrent et $\sim_I$, la congruence induite par $I = \Sigma^2 - D$. Alors, $\Sigma^\star/_{\sim_I}$, le quotient du monoïde libre $\Sigma^\star$ par la congruence $\sim_I$, est le monoïde libre, partiellement commutatif, induit par la relation I. Plus simplement, il est appelé le* monoïde de traces *et noté $\mathbb{M}(\Sigma, D)$.*

Les éléments de $\mathbb{M}(\Sigma, D)$, qui sont les classes d'équivalence de $\Sigma^\star$ par la relation $\sim_I$, sont appelés des *traces*. Tout sous-ensemble $L \subseteq \mathbb{M}(\Sigma, D)$ est appelé langage de traces.

Par définition d'une classe d'équivalence, à chaque trace $[u]_{\sim_I} \in \mathbb{M}(\Sigma, D)$ est associé un ensemble de mots $\{v \mid u \sim_I v\} \subseteq \Sigma^\star$, clos par $\sim_I$. De même, à tout ensemble de mots équivalents par $\sim_I$, correspond une trace. Ainsi, cet isomorphisme nous permet d'assimiler un langage de traces L à l'ensemble $\{u \in \Sigma^\star \mid [u]_{\sim_I} \in L\}$ des mots clos par $\sim_I$ qu'il représente, noté $\bigcup L$. Il existe donc un isomorphisme permettant de passer d'un langage d'un $\mathbb{M}(\Sigma, D)$-automate à un sous-ensemble de $\Sigma^\star$ clos par commutation pour la relation $\sim_I$. Dans ce cadre, un langage de traces $L \subseteq \mathbb{M}(\Sigma, D)$ est reconnaissable dans $\mathbb{M}(\Sigma, D)$, si l'ensemble $\bigcup L$ des mots qu'il représente est régulier.

En respectant l'hypothèse que nous avons posée au début de ce chapitre, nous définissons maintenant les spécifications des systèmes parallèles interagissant par mémoire partagée comme étant des expressions rationnelles de $\mathbb{M}(\Sigma, D)$ (ce qui est noté $\mathrm{REX}(\mathbb{M}(\Sigma, D))$). De plus, étant donné une expression rationnelle α de $\mathrm{REX}(\mathbb{M}(\Sigma, D))$, nous notons $\mathcal{L}_{\mathbb{M}(\Sigma,D)}(\alpha) \subseteq \mathbb{M}(\Sigma, D)$ le langage de traces de α et $\mathcal{L}_{\Sigma^\star}(\alpha) \subseteq \Sigma^\star$ le langage de mots clos par $\sim_I$ engendré par $\bigcup \mathcal{L}_{\mathbb{M}(\Sigma,D)}(\alpha)$.

1.2.2 Reconnaissables et corationnels

Nous nous intéressons maintenant aux modèles opérationnels que l'on peut définir dans le monoïde de traces. L'idée première est de travailler sur les ensembles réguliers de mots de $\Sigma^\star$, c'est-à-dire de choisir comme modèle d'exécution le même modèle d'exécution que pour les systèmes séquentiels. Cependant, ce choix n'est pas forcément le plus adapté. En effet, nous

donnons, dans la partie suivante, un modèle moins expressif mais qui possède de meilleures propriétés, appelé automates asynchrones.

Tout d'abord, il n'est pas possible de décider si un langage de traces est régulier, c'est-à-dire si une spécification rationnelle possède un modèle opérationnel donné sous forme d'automate. En effet, Sakarovitch a montré le résultat suivant dans [Sakarovitch 92].

Théorème 1.10 ([Sakarovitch 92]) *Soient (Σ, D), un alphabet concurrent et α, une expression rationnelle dans $REX(\mathbb{M}(\Sigma, D))$. Savoir si $\bigcup \mathcal{L}_{\mathbb{M}(\Sigma,D)}(\alpha)$ est régulier est décidable si, et seulement si, la fermeture réflexive de $I = \Sigma^2 - D$ est transitive. Ainsi, dans le cas général, ce problème est indécidable.*

Cependant, Ochmanski a montré dans [Ochmanski 85] qu'il existe une sous-classe syntaxique des expressions rationnelles du monoïde de traces, appelée traces corationnelles, dont les éléments ont un langage reconnaissable. De plus, Muscholl et Peled ont montré dans [Muscholl 99] que cette sous-classe était décidable : plus précisément, savoir si une trace est corationnelle est coNP-complet.

Définition 1.11 (corationnalité) *Soient (Σ, D), un alphabet concurrent et $D = \Sigma^2 - I$. Un mot $u \in \Sigma^\star$ est D-*connexe *si le graphe $(\Sigma(u), D)$, dont les sommets sont les lettres de u, est connexe. Une expression rationnelle α est dite (D-)*corationnelle *("star-connected" en anglais) si, pour chaque sous-expression $\beta^\star$ de α, tout $u \in \bigcup \mathcal{L}_{\Sigma^\star}(\beta)$ est une trace D-connexe.*

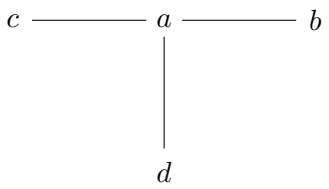


FIG. *1.1 – Graphe associé au mot $u = (cb)^\star aac(ca + d)^\star$, dans le monoïde de traces $\mathbb{M}(\Sigma, I)$, où $\Sigma = \{a, b, c, d\}$ et $I = \{(c, b), (b, c), (c, d), (d, c), (b, d), (d, b)\}$). u n'est pas corationnelle car cd n'est pas D-connexe.*

La figure 1.1 illustre la construction du graphe $(\Sigma(u), \Sigma(u) \times \Sigma(u) \cap D)$ associé au mot $u = (cb)^\star aac(ca + d)^\star$ et à la relation d'indépendance $I = \{(c, b), (b, c), (c, d), (d, c), (b, d), (d, b)\}$. Le graphe restreint à $\{c, b\}$ n'étant pas connexe, (cb) n'est pas D-connexe, et donc $u^\star$ n'est pas une expression corationnelle.

Voici maintenant le résultat d'Ochmanski concernant les traces corationnelles. Ce théorème indique que celles-ci forment un sous-ensemble strict des traces rationnelles et correspondent à l'ensemble des reconnaissables du monoïde de traces.

Théorème 1.12 ([Ochmanski 85]) *Soient $L \subseteq \mathbb{M}(\Sigma, D)$, un langage de traces et $I = \Sigma^2 - D$. Alors, les propositions suivantes sont équivalentes :*
- *L est reconnaissable dans $\mathbb{M}(\Sigma, D)$;*
- *$\bigcup L$ est régulier dans $\Sigma^\star$;*
- *il existe une expression corationnelle α de $REX(\mathbb{M}(\Sigma, D))$ telle que $\mathcal{L}_{\mathbb{M}(\Sigma,D)}(\alpha) = L$;*

– *il existe une expression corationnelle* α *de* $REX(\Sigma^\star)$ *telle que* $[\mathcal{L}_{\Sigma^\star}(\alpha)]_{\sim_I} = \bigcup L$.

Remarquons qu'il existe, en général, une infinité d'expressions rationnelles qui permettent de décrire le même ensemble rationnel. Pour avoir la régularité, il suffit qu'une seule d'entre elles soit corationnelle, ce qui est un problème indécidable. Par exemple, pour l'alphabet $\Sigma = \{a, b\}$ et la relation d'indépendance $I = \{(a, b), (b, a)\}$, les expressions rationnelles $\alpha_1 = (ab + a + b)^\star$ et $\alpha_2 = (a + b)^\star$ décrivent le même langage $\{a, b\}^\star$. Cependant, α_1 n'est pas corationnelle, alors que α_2 l'est.

En résumé, les théorèmes 1.10 et 1.12 semblent indiquer que les $\Sigma^\star$-automates ne sont pas les bons modèles d'exécution qui correspondent aux rationnels de $\mathbb{M}(\Sigma, D)$: en effet, étant donné une spécification rationnelle, il est, d'une part, impossible de savoir s'il existe un $\Sigma^\star$-automate qui "simule" le comportement de cette spécification. D'autre part, tous les $\Sigma^\star$-automates ne correspondent pas à des modèles d'exécution : en effet, il faut en plus que le langage généré soit clos par $\sim_I$, ce qui n'est pas décidable. Pour résoudre ces problèmes, le pouvoir d'expression du langage de spécification doit être limité en considérant uniquement les traces corationnelles : dans ce but nous introduisons dans la suite une sous-classe des automates, les automates asynchrones. Nous montrons, dans la partie suivante, que, comme dans le cas des modèles séquentiels, ces modèles ont un pouvoir d'expression équivalent.

1.2.3 Réseaux d'automates asynchrones

Dans la partie précédente, nous avons indiqué qu'une expression rationnelle dans le monoïde de traces ne correspondait pas forcément à un modèle opérationnel exprimé à l'aide d'un automate. Cependant, une sous-classe syntaxique a été identifiée, composée des corationnels, dont les comportements que l'on peut spécifier correspondent toujours à un modèle d'exécution. Nous nous intéressons, maintenant, à la classe des modèles d'exécution qui correspondent à une spécification corationnelle. Cette classe est donc incluse dans celle des automates. Ce modèle d'exécution plus adapté a été proposé par Zielonka dans [Zielonka 87] et il correspond à un réseau de systèmes séquentiels à événements discrets qui interagissent par synchronisation sur certains événements (et donc par mémoire partagée).

Définition 1.13 (réseau d'automates asynchrones) *Soient* $\mathcal{P}$*, un ensemble fini de processus et* $\Sigma = \bigcup_{p\in\mathcal{P}} \Sigma_p$*, l'union d'une famille d'alphabets* Σ_p *non nécessairement disjoints. Etant donné un produit* $Q^1 \times \ldots Q^{|\mathcal{P}|}$ *d'ensembles d'états associés à chaque processus, un* réseau de $\Sigma^\star$-automates asynchrones *est une structure* $\mathcal{R} = \langle S_i, (\mathcal{A}_l)_{l\in 2^\mathcal{P}}, S_f\rangle$ *où :*

– *pour tout* $l \in 2^\mathcal{P}$*,* $\mathcal{A}_l$ *est un* $\Sigma^\star$*-automate* $(S^l, \rightarrow_l, \Sigma^l, S^l_i, S^l_f)$ *qui peut lire et modifier les états des processus contenu dans* l*, c'est-à-dire avec* $S^l = Q^{k_1} \times \ldots \times Q^{k_{|l|}}$ *pour* $\{k_1 < \ldots < k_{|l|}\} = l$ *et* $\Sigma^l = \bigcup_{p\in l} \Sigma_p$ *;*

– S_i *est un ensemble d'états initiaux globaux, c'est-à-dire* $S_i \subseteq (S^1_i \times \ldots \times S^{|\mathcal{P}|}_i)$ *;*

– S_f *est un ensemble d'états finaux globaux, c'est-à-dire* $S_f \subseteq (S^1_f \times \ldots \times S^{|\mathcal{P}|}_f)$.

Un état global $s \in S$ *est un* $|\mathcal{P}|$*-uplet d'états locaux, c'est-à-dire* $S = Q^1 \times \ldots \times Q^{|\mathcal{P}|}$*. La* relation de transition globale*, notée* $\rightarrow \subseteq S \times \Sigma \times S$ *est définie comme suit :* $(q^p)_{p\in\mathcal{P}} \xrightarrow{\sigma} (q'^p)_{p\in\mathcal{P}}$ *si* $(q^p)_{p\in l} \xrightarrow{\sigma}_l (q'^p)_{p\in l}$ *pour* $l = \{p \mid \sigma \in \Sigma_p\}$ *et* $q^p = q'^p$ *sinon. L'automate global associé à* $\mathcal{R}$ *est noté* $\langle S_i, \bigotimes_{l\in\mathcal{P}} \mathcal{A}_l, S_f\rangle$ *(*$\otimes$ *désignant le produit asynchrone des automates locaux).*

Un réseau d'automates asynchrones est dit *déterministe* si $\rightarrow$ est une fonction de $S \times \Sigma$ dans S et S_i est un singleton. Un réseau d'automates asynchrones est à états initiaux locaux si $S_i = S_i^1 \times \ldots \times S_i^{|\mathcal{P}|}$ et à états finaux locaux, si $S_f = S_f^1 \times \ldots \times S_f^{|\mathcal{P}|}$. Remarquons qu'un réseau d'automates asynchrones déterministe est forcément à états initiaux locaux. De plus, l'automate global est noté $\langle \otimes_{l \in 2^{\mathcal{P}}} \mathcal{A}_l, S_f \rangle$, $\langle S_i, \otimes_{l \in 2^{\mathcal{P}}} \mathcal{A}_l \rangle$ ou $\otimes_{l \in 2^{\mathcal{P}}} \mathcal{A}_p$ si le réseau est, respectivement, à états initiaux locaux, à états finaux locaux ou à états initiaux et finaux locaux. En effet, cette notation indique que les états initiaux ou finaux globaux peuvent être alors reconstruits à partir du produit cartésien des états initiaux ou finaux locaux. Enfin, un automate asynchrone est dit *sans blocage* si tous les états globaux accessibles sont aussi co-accessibles (c'est-à-dire qu'il existe un chemin allant de l'état courant vers un état final). Le langage d'un réseau d'automates asynchrones $\mathcal{A} = \langle S_i, (\mathcal{A}_l)_{p \in 2^{\mathcal{P}}}, S_f \rangle$, noté $\mathcal{L}_{\Sigma^\star}(\langle S_i, \otimes_{l \in 2^{\mathcal{P}}} \mathcal{A}_l, S_f \rangle$, est l'ensemble des mots $u = a_1 \ldots a_{|u|}$ tels qu'il existe un chemin $S_0 \xrightarrow{a_1} \ldots \xrightarrow{a_{|u|}} S_{|u|}$ acceptant dans $\langle S_i, \otimes_{l \in 2^{\mathcal{P}}} \mathcal{A}_l, S_f \rangle$.

Nous donnons maintenant le résultat principal concernant l'équivalence entre l'expressivité des spécifications corationnelles et des réseaux d'automates asynchrones. Plus précisément, Zielonka a montré dans [Zielonka 87] que tout langage régulier et clos par une congruence $\sim_I$ est équivalent à un réseau d'automates asynchrones déterministes. Et, il a démontré dans [Zielonka 89] que le même résultat pouvait se prouver dans le cas où les réseaux sont sans blocage (mais non déterministes) (voir [Baudru 05] pour une preuve constructive de ce résultat). Combinés au théorème 1.12, ces résultats permettent d'énoncer le théorème suivant :

Théorème 1.14 ([Zielonka 87, Zielonka 89]) *Soient (Σ, D), un alphabet concurrent et $L \subseteq \mathbb{M}(\Sigma, D)$, un langage de traces. Alors, les trois affirmations suivantes sont équivalentes.*
- *L est le langage d'une expression corationnelle ;*
- *L est le langage d'un réseau déterministe d'automates asynchrones ;*
- *L est le langage d'un réseau sans blocage d'automates asynchrones à états finaux locaux.*

Notons, d'une part, que la traduction d'un modèle vers un autre est plus complexe que pour les modèles séquentiels. En effet, elle se fait au moins en taille doublement exponentielle ([Baudru 05, Genest 06b]).

D'autre part, contrairement aux automates, il n'y a pas un unique modèle de bas niveau. En effet, la figure 1.2 donne un exemple de réseau sans blocage d'automates asynchrones à états finaux locaux qui ne peut pas être transformé en réseau déterministe d'automates asynchrones. Le réseau d'automates asynchrones représenté sur cette figure a deux localités p_1 et p_2. L'automate de gauche est associé à la localité p_1, l'automate du milieu à p_2 et l'automate de droite aux localités $\{p_1, p_2\}$. Le monoïde sous-jacent est $\mathbb{M}(\{a, b, c\}, \{(a, c), (c, a)\})$. Le réseau représenté reconnaît le même langage que l'expression corationnelle $(a + c)^\star b(aa^\star + cc^\star)$, avec $I = \{(a, c), (c, a)\}$.

Finalement, nous donnons les résultats de complexité pour les mêmes problèmes de décision évoqués dans la partie précédente de ce chapitre. Ces résultats amènent deux commentaires par rapport aux résultats énoncés sur les modèles séquentiels. Le premier est que la plupart des problèmes sont indécidables. Le second est que la complexité dépend de la représentation (de bas niveau, ou de haut niveau) choisie.

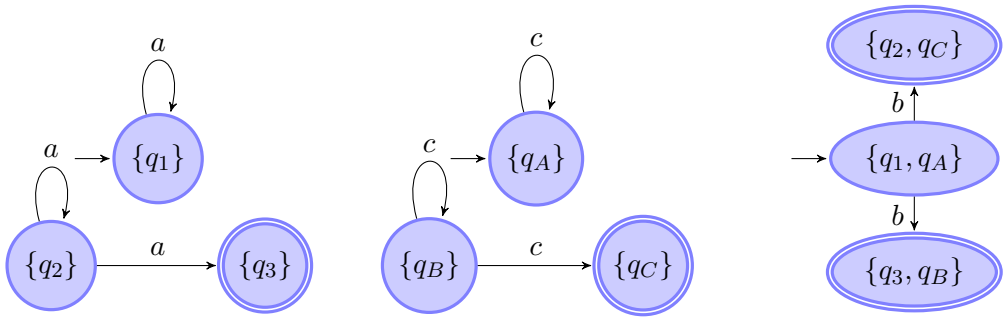


FIG. *1.2 – Un exemple de* $\{a,b\} \cup \{b,c\}^\star$*-automates asynchrones qui reconnaît le même langage que l'expression corationnelle* $(a+c)^\star b(aa^\star + cc^\star)$*, avec* $I = \{(a,c),(c,a)\}$.

Proposition 1.15 *Pour les systèmes asynchrones, nous avons les résultats de complexité suivants, triés verticalement par problème et horizontalement par générateur des langages L et L′ : (RAA désignant les réseaux d'automates asynchrones) :*

	$REX(\mathbb{M}(\Sigma, D))$	*L corationnel*	*RAA*	*RAA déterministes*
$x \in L$	*NP-c*	*NP-c*	*NP*	*NLOGSPACE-c*
$L \cap L' = \emptyset$	*Indécidable*	*PSPACE-c*	*PSPACE-c*	*NLOGSPACE-c*
$L' \subseteq L$	*Indécidable*	*EXPSPACE-c*	*EXPSPACE*	*NLOGSPACE-c*

Preuve: Les preuves d'indécidabilité proviennent de [Diekert 95]. Les preuves de complétude viennent directement des résultats présentés dans la proposition 1.8 et de l'existence d'un automate de taille exponentielle qui reconnait la fermeture par $\sim_I$ des langages considérés (th. 1.14). De plus, il faut remarquer que $[L]_{\sim_I} \cap [L']_{\sim_I} = \emptyset$ si, et seulement si, $L \cap [L'] = \emptyset$ et $[L] \subseteq [L']$ si, et seulement si, $\mathcal{C}_{\Sigma^\star}(L) \cap [L'] = \emptyset$, avec $\mathcal{C}_{\Sigma^\star}(L)$, le complément de L dans $\Sigma^\star$.

Pour les automates, la complémentation donne une structure de taille exponentielle. Pour les réseaux asynchrones, [Klarlund 94] a montré que les réseaux d'automates asynchrones peuvent se déterminiser en un réseau de taille doublement exponentielle, ce qui donne l'appartenance à EXPSPACE.

A notre connaissance, caractériser la complexité exacte des questions de l'appartenance et de la vacuité de l'intersection pour les réseaux d'automates asynchrones est un problème ouvert. □

1.3 Modèles communicants

Dans les parties précédentes, nous avons introduit les définitions et les résultats principaux concernant les modèles séquentiels, d'une part, et parallèles et asynchrones, d'autre part. Nous continuons l'exploration des modèles parallèles dans cette partie, en nous concentrant, cette fois, sur les systèmes communiquant à l'aide d'échanges de messages asynchrones. Dans cette partie, nous présentons tout d'abord, dans la partie 1.3.1, un modèle d'ordres partiels étiquetés, appelé pomset (pour "partially ordered multi-set"). Ce modèle est très général et permet de modéliser une large classe de systèmes parallèles et répartis.

Dans la partie 1.3.2, nous restreignons, ensuite, cette expressivité pour présenter les HMSC, pour "High-level Message Sequence Charts", un langage de spécification normalisé fondé sur les pomsets et utilisé pour décrire des systèmes parallèles communiquant par échange asynchrone de messages. Ensuite, dans la partie 1.3.3, nous présentons les CHMSC, une classe de langage de spécification qui étend celle des HMSC, et nous donnons des sous-classes de ces modèles de spécification qui sont équivalentes à des modèles d'exécution donnés sous forme d'automates. Pour terminer, la partie 1.3.4 présente un modèle d'exécution pour ces systèmes communicants, et donne les liens qui relient des sous-classes syntaxiques des langages de spécification présentés tout au long de cette partie avec ce modèle. Ainsi, comme dans le cadre séquentiel ou asynchrone, il existe des formalismes de haut niveau et de bas niveau ayant un pouvoir d'expression équivalent.

1.3.1 Pomsets

Les définitions qui vont suivre s'inspirent fortement de celles de Pratt [Pratt 86] et Gisher [Gischer 88], qui ont introduit le formalisme des pomsets à la fin des années 1980. La théorie des pomsets est le point de rencontre entre la théorie des langages formels, où les séquences de lettres peuvent être vues comme des pomsets dont l'ordre est total, et la théorie des ordres partiels qui peuvent être vus comme des pomsets à étiquetage bijectif.

Plus formellement, la définition suivante introduit la notion d'ordre partiel.

Définition 1.16 (LPO) *Etant donné un alphabet Σ, un* LPO *(pour "labeled partial order") est un ensemble E d'événements étiquetés partiellement ordonnés. Plus précisément, c'est une structure $(E, \leq, \lambda)$ où $\leq \subset E \times E$ est une relation binaire, réflexive, anti-symétrique et transitive, et $\lambda : E \to \Sigma$ étiquette chaque élément de E par une action de Σ.*

Quand nécessaire, nous noterons $(E_l, \leq_l, \lambda_l)$ les différents éléments du LPO l. La relation de couverture (appelée ausi réduction transitive), notée $\prec$, est la plus petite relation dont la fermeture transitive est égale à $\leq$. Le graphe $(E_l, \prec)$ est appelé diagramme de Hasse de l.

Définition 1.17 (pomset) *Un* isomorphisme *de* LPO *est un isomorphisme d'ordres partiels qui préserve l'étiquetage. Plus formellement, $f : (E_1, \leq_1, \lambda_1) \to (E_2, \leq_2, \lambda_2)$ est un isomorphisme de* LPO *si c'est une fonction bijective de E_1 dans E_2 et si, pour tout e et e' dans E_1, $e \leq_1 e'$ est équivalent à $f(e) \leq_2 f(e')$ et $\lambda_1(e) = \lambda_2(f(e))$. Un* pomset *est la classe d'un ensemble isomorphique de* LPO*. On notera $[E, \leq, \lambda]$ le pomset qui représente la classe du* LPO *$(E, \leq, \lambda)$, et $[E_p, \leq_p, \lambda_p]$ les composantes du pomset p. Plus généralement, nous notons $[l]$ la classe d'isomorphisme du* LPO *l.*

De manière intuitive, les pomsets se focalisent principalement sur les étiquettes et sur la manière dont elles sont ordonnées. Le pomset vide est noté ε. De plus, étant donné un pomset p, on appelle *représentant* de p un LPO l tel que $[l] = p$ (c'est-à-dire un représentant de la classe d'isomorphisme induite par le pomset sur les LPO).

Nous donnons maintenant quelques définitions techniques, qui nous seront utiles par la suite. Tout d'abord, un pomset p est "*auto-concurrent*" si, il existe deux événements e_1 et e_2 de E_p tels que $e_1 \not\leq_p e_2$, $e_2 \not\leq_p e_1$ et $\lambda_p(e_1) = \lambda_p(e_2)$. Ensuite, la *restriction* d'un pomset p à un

ensemble d'événements E, notée $p_{|E}$, est définie comme suit : $p_{|E} = [E_p \cap E, \leq_p \cap E \times E, \lambda_{p|E}]$, où $\lambda_{p|E}$ est la restriction de λ_p au domaine E. De plus, la *projection* d'un pomset p étiqueté par Σ, sur un alphabet observable $\Sigma_o \subseteq \Sigma$, est une fonction π_{Σ_o} qui restreint p à ses éléments observables. Plus formellement, $\pi_{\Sigma_o}(p) = p_{|\lambda^{-1}(\Sigma_o)}$. L'idéal d'un événement e dans un pomset p, noté $I_p(e)$, est la fermeture par précédence de cet événement dans p. Plus précisément $I_p(e) = \{e' \in E_p \mid e' \leq_p e\}$. De plus, le nombre d'événements de p étiquetés par l'action a est noté $|p|_a$.

Ensuite, nous appellons *extension linéaire* d'un pomset p, un pomset l dont les ensembles d'événements sont isomorphes, et dont la relation d'ordre $\leq_l$ étend celle de p, (c.a.d. $\leq_p \subseteq \leq_l$) et est totale (c'est-à-dire $\forall (e, e') \in E_p$, $e \leq_l e'$ ou $e' \leq_l e$). Une extension linéaire $e_1 \prec \ldots \prec e_n$ d'un pomset p, étiqueté par Σ, peut être vue comme un mot $u = \lambda_p(e_1) \ldots \lambda_p(e_n)$ de $\Sigma^\star$. L'ensemble des mots induits par les extensions possibles d'un pomset p est appelé le langage de linéarisations de p et est noté $Lin(p)$.

La figure 1.3 donne un exemple de pomset étiqueté par $\Sigma = \{a, b\}$, dont le langage de linéarisation est $\{baab; baba\}$. Les événements sont représentés par des cercles contenant l'étiquette de l'événement. Comme nous travaillons sur des classes d'isomorphisme, les événements ne sont pas nommés. Les relations de causalités sont représentées par des flèches. Afin de simplifier les figures, seul le diagramme de Hasse des pomsets (la relation $\prec$) est représenté.

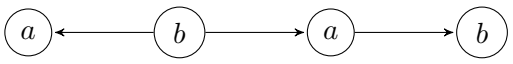


FIG. *1.3 – Un exemple de pomset pour lequel* $\Sigma = \{a, b\}$*, dont le langage de linéarisation est* $\{baab; baba\}$.

Nous introduisons maintenant un opérateur de composition de pomsets, le but étant d'obtenir un monoïde afin d'y appliquer les résultats énoncés dans les parties précédentes.

Définition 1.18 (composition de pomsets) *Soient* Σ*, un alphabet,* $D \subseteq \Sigma^2$*, une relation,* p_1 *et* p_2*, deux pomsets étiquetés par* Σ *et* l_1 *et* l_2*, des* LPO *les représentant avec des ensembles* distincts *d'événements. La* composition *de* p_1 *et* p_2*, notée* $p_1 \circledcirc_D p_2$*, est un opérateur qui fait l'union de* l_1 *et* l_2*, qui ajoute des causalités entre les paires d'événements* $(e, e') \in E_{l_1} \times E_{l_2}$ *telles que* $(\lambda_{l_1}(e), \lambda_{l_2}(e')) \in D$*, et qui prend la classe d'isomorphisme du résulat. Plus formellement,* $p_1 \circledcirc_D p_2 = [E_{l_1} \uplus E_{l_2}, (\leq_{l_1} \cup \leq_{l_2} \cup \{(e, e') \in E_{l_1} \times E_{l_2} \mid (\lambda_{l_1}(e), \lambda_{l_2}(e')) \in D\})^*, \lambda_{l_1} \cup \lambda_{l_2}]$.

Nous notons $\mathbb{P}(\Sigma, D)$, l'ensemble des pomsets étiquetés par Σ et muni de la relation de composition $\circledcirc_D$. La définition précédente s'inspire de la définition de la composition séquentielle locale définie dans les travaux de Pratt [Pratt 86]. Le choix de D permet d'exprimer différents opérateurs classiques de composition. Ainsi, la composition parallèle correspond à $D = \emptyset$, la composition synchrone (appelée aussi composition séquentielle forte) correspond à $D = \Sigma_P^2$. La composition de pomsets est illustrée dans la figure 1.4, où $D = \{(a, a), (c, c), (a, c), (c, a)\}$ et les causalités ajoutées par les compositions sont représentées en traits pointillés.

Il est facile de montrer que $\circledcirc_D$ est une loi de composition stable et associative. Ainsi, nous obtenons la proposition suivante, qui nous permet de retomber dans le cadre formel

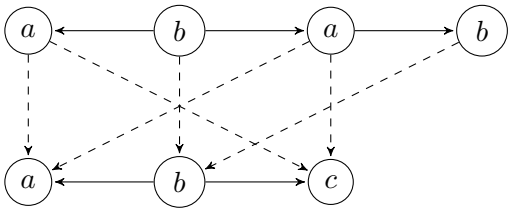


FIG. *1.4 – Un exemple de composition de pomsets. Ici, $D = \{(a,a),(b,b),(c,c),(a,c),(c,a)\}$ et les causalités ajoutées sont représentées en traits pointillés.*

introduit depuis le début de ce chapitre.

Proposition 1.19 *Pour tout alphabet Σ et toute relation $D \subseteq \Sigma^2$, $(\mathbb{P}(\Sigma, D), \circledcirc_D, \varepsilon)$ est un monoïde qui sera simplement noté $\mathbb{P}(\Sigma, D)$.*

Cependant, $\mathbb{P}(\Sigma, D)$ ne possède généralement aucune propriété particulière (il n'est ni libre, ni finiment engendré, ni reconnaissable, etc.). Dans la suite de ce chapitre, nous identifions et donnons les propriétés de certaines sous-classes rationnelles de $\mathbb{P}(\Sigma, D)$ que nous utilisons, par la suite, comme langages de spécification pour les systèmes communicants.

1.3.2 High-level Message Sequence Charts (HMSC)

Les MSC (ou "Message Sequence Charts") ont rencontré un intérêt considérable pendant cette dernière décennie. C'est un langage de spécification graphique normalisé par l'ITU ("International Telecommunication Union"), qui publia, en 1993, la recommandation Z.120, où est définie la syntaxe des MSC, mais pas leur sémantique. Ce formalisme permet d'exprimer simplement des ensembles finis d'interactions asynchrones entre entités communicantes. A la même date, Philips propose un nouveau langage de spécification graphique, appelé *Interworking* ([Mauw 93]), qui ressemble beaucoup aux MSC, bien qu'ayant une sémantique synchrone fondée sur les algèbres de processus. En 1994, Mauw et Reniers s'inspirent de ce travail pour donner dans [Mauw 94] une sémantique asynchrone aux MSC qui est intégrée en 1995 par l'ITU dans la norme Z.120. L'année suivante, l'ITU étend la syntaxe des MSC à l'aide d'opérateurs de composition (séquentielle et parallèle), afin de pouvoir exprimer des ensembles infinis de MSC. Ce nouveau langage de spécification s'appelle HMSC (pour "High-level MSC") et, comme pour les MSC, sa sémantique est donnée quelques années plus tard par Mauw et Reniers dans [Mauw 97, Reniers 98], puis intégrée à la recommandation Z.120 pour donner le document [ITUTS 99]. En 2003, la version 2.0 d'UML [OMG 03] intègre les "Diagrammes de Séquence". Sous un nom différent, on retrouve le noyau syntaxique des HMSC.

Dans la suite de cette partie, nous présentons une sémantique des HMSC fondée sur des expressions rationnelles construites à l'aide de familles restreintes d'ordres partiels. Bien que la sémantique initiale des HMSC ait été donnée en utilisant des algèbres de processus, l'approche sémantique fondée sur des modèles d'ordres partiels est communément adoptée dans le domaine. Se reporter par exemple à [Muscholl 98, Muscholl 99, Alur 99, Hélouët 00b, Genest 02b, Morin 02, Henriksen 05].

Nous commençons par formaliser la notion d'entité communicante (appelée aussi processus, agent, localité, ...) et de media de communication. Pour cela, fixons un ensemble $\mathcal{P}$ de

localités, un ensemble $\mathcal{M}$ de messages et un ensemble $\mathcal{I}$ d'actions internes. Nous définissons ensuite les ensembles d'actions suivants :

- $\Sigma_! = \{p!q(m) \mid p, q \in \mathcal{P}, m \in \mathcal{M}\}$, $p!q(m)$ signifiant "p envoie un message m à q" ;
- $\Sigma_? = \{p?q(m) \mid p, q \in \mathcal{P}, m \in \mathcal{M}\}$, $p?q(m)$ signifiant "p réceptionne un message m provenant de q" ;
- $\Sigma_i = \{p(a) \mid p \in \mathcal{P}, a \in \mathcal{I}\}$, $p(a)$ signifiant "p effectue l'action interne a" ;
- $\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}} = \Sigma_! \cup \Sigma_? \cup \Sigma_i$ est l'alphabet global.

La localité associée à une action, correspondant au nom du processus qui effectue l'action considérée, est une fonction, notée $loc : \Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}} \to \mathcal{P}$, définie comme suit : $loc(p!q(m)) = loc(p?q(m)) = loc(p(a)) = p$. Plus généralement, nous notons $\Sigma_p = \{\sigma \mid loc(\sigma) = p\}$ et si $o = [E, \leq, \lambda]$ est un pomset, pour tout $p \in \mathcal{P}$, $E_p = E \cap \lambda^{-1}(\Sigma_p)$. L'alphabet $\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$, muni de la fonction loc, est appelé un *alphabet localisé*. Etant donné un pomset o, nous définissons la relation de causalité, $\ll \subseteq (E \times E)$, induite par l'appariement de la réception d'un message avec l'envoi correspondant, de la manière suivante : $e \ll e'$ si il existe $p, q \in \mathcal{P}$ et $m \in \mathcal{M}$ tels que $\lambda(e) = p!q(m)$, $\lambda(e') = q?p(m)$ et $|I_o(e)|_{p!q(m)} = |I_o(e')|_{q?p(m)}$, c'est-à-dire si le nombre d'envois et de réceptions de messages étiquetés par m, dans le passé causal des deux événements, est le même. Ainsi, par définition de l'appariement, deux messages de même type, utilisant un même canal ne peuvent pas se croiser. Cette politique de routage est appelée "weak-FIFO".

Nous donnons maintenant la définition d'un MSC, qui est le modèle qui va nous servir de brique de base pour construire des spécifications de systèmes communicants.

Définition 1.20 (MSC) *Soit $\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$, un alphabet localisé. Un* MSC *(pour "Message Sequence Charts" en anglais) est un pomset $o = [E, \leq, \lambda]$ étiqueté par $\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$ tel que :*

(i) tous les événements étant sur la même localité sont totalement ordonnés, c'est-à-dire que, $\forall e, e' \in E$, si $loc(e) = loc(e')$ alors $e \leq e'$ ou $e' \leq e$;

(ii) o est clos par communication, c'est-à-dire que, pour tous $p, q \in \mathcal{P}$ et $m \in \mathcal{M}$, $|o|_{p!q(m)} = |o|_{q?p(m)}$;

(iii) o ne rajoute pas d'autre ordre que la causalité locale et celle des messages, c'est-à-dire que la relation $\leq$ est exactement $(\ll \bigcup_{p \in \mathcal{P}} (\leq \cap E_p^2))^*$.

Nous notons $\mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$, le monoïde des MSC *muni de la loi de composition interne $\circ = \circledcirc_{D_{loc}}$, avec $D_{loc} = \{(a, b) \in \Sigma^2 \mid loc(a) = loc(b)\}$ et dont l'élément neutre est le pomset vide ε.*

De manière intuitive, un MSC est naturellement associé à une exécution d'un système parallèle communiquant par échange de messages : chaque processus effectue des actions de manière séquentielle (imposée par l'ordre total) et lorsqu'un processus envoie un message vers un autre processus, il peut continuer à exécuter des actions sans avoir à attendre que le message envoyé soit réceptionné. L'ordre (localement) total est conservé lors de la composition. Cependant, un MSC ne décrit qu'un comportement fini. Afin de disposer d'un langage de spécification suffisamment expressif, nous avons besoin de rajouter des opérateurs de choix et d'itération. En d'autres termes, nous avons besoin d'expressions rationnelles de MSC.

Définition 1.21 (HMSC) *Un* HMSC *est une expression rationnelle de $\mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$.*

Etant donné un HMSC α, nous notons $\Gamma(\alpha)$ l'ensemble des MSC qui apparaissent dans l'expression rationnelle α et $\mathcal{L}_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})}(\alpha)$ l'ensemble des MSC qu'il engendre. Nous pouvons remarquer, ensuite, qu'une expression rationnelle de $\mathrm{REX}(\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I}))$ peut être interprétée comme une expression rationnelle de $\mathrm{REX}(\Gamma(\alpha)^\star)$, en considérant le nom des MSC plutôt que les morceaux d'ordre qu'ils définissent. Ainsi, nous pouvons associer à tout HMSC α, un $\Gamma(\alpha)^\star$-automate $\mathcal{A}_\alpha$ en utilisant par exemple l'algorithme de Thompson ([Thompson 68]) permettant de passer d'une expression régulière à un automate possédant les mêmes propriétés structurelles. En particulier, tout MSC m, appartenant au langage $\mathcal{L}_{\mathbb{MSC}(\mathcal{M},\mathcal{P},\mathcal{I})}(\beta)$ d'une sous-expression $\beta^\star$ de α, correspondra à un cycle étiqueté par m dans $\mathcal{A}_\alpha$. De plus, l'automate produit à partir d'une expression rationnelle n'aura pas de transitions étiquetées par ε. De manière quelque peu abusive, nous dirons alors que $\mathcal{A}_\alpha$ est structurellement semblable à α.

Afin d'identifier des sous-classes syntaxiques reconnaissables de $\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})$, nous introduisons la notion de graphe de communication. Le graphe de communication d'un MSC m est le graphe $m/_{\sim_{loc}}$, quotient de m par la relation d'équivalence $\sim_{loc}$ définie comme suit : $e \sim_{loc} e'$ si $loc(e) = loc(e')$. Un MSC est dit *connexe* si son graphe de communication est connexe.

La figure 1.5 illustre cette notion sur un exemple. A gauche de cette figure, un MSC est représenté. Les localités sont représentées par des lignes verticales qui ordonnent totalement les éléments qui s'y trouvent, du haut vers le bas. Les messages sont représentés surmontés de leur nom et les actions internes apparaissent comme des cercles noirs. Par exemple, le premier événement sur le processus p est étiqueté $p!q(m)$ et le premier événement sur l'instance r est étiqueté $r(a)$.

A droite de la figure 1.5, est représenté le graphe de communication associé : chaque noeud de ce graphe représente un processus de $\mathcal{P}$, et il existe un arc orienté entre deux noeuds si il existe une communication entre ces deux processus dans c.

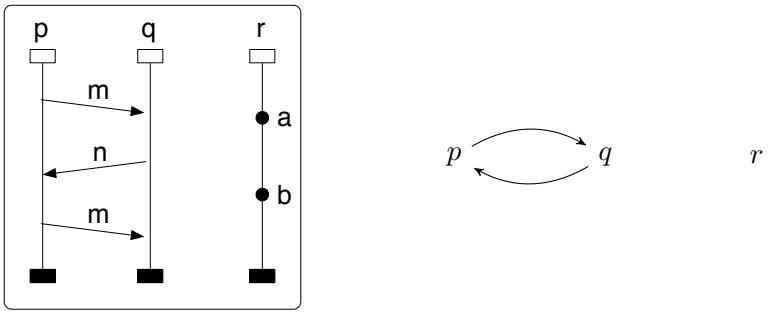


FIG. *1.5 – Un* MSC *dans* $\mathbb{MSC}(\{p,q,r\},\{m,n\},\{a,b\})$ *et son graphe de communication.*

Régularité

Nous nous intéressons, maintenant, aux sous-ensembles de $\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})$ dont le langage de linéarisation est régulier. De tels ensembles sont abusivement dits réguliers : c'est de leur langage de linéarisation qu'il est question. Ces sous-ensembles correspondent aux

spécifications de modèles parallèles dont le comportement peut être simulé par un modèle séquentiel d'exécution. Tout d'abord, Muscholl et Peled ont montré dans [Muscholl 99] qu'il était indécidable de savoir si un HMSC engendrait un langage régulier de linéarisation. Cependant, ils ont montré qu'il existait une propriété syntaxique dont l'algorithme de test est CoNP-complet qui permet d'assurer que le langage généré est régulier.

Théorème 1.22 ([Henriksen 05]) *Soit $L \subseteq \mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$, un langage finiment engendré. Les deux affirmations suivantes sont équivalentes :*

- *$Lin(L)$ est régulier dans $\Sigma^\star$;*
- *il existe un HMSC α tel que $\mathcal{L}_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})}(\alpha) = L$ et pour toute sous-expression $\beta^\star$ de α, le graphe de communication de tout MSC appartenant à $\mathcal{L}_{\mathbb{MSC}(\mathcal{M},\mathcal{P},\mathcal{I})}(\beta)$ est fortement connexe.*

Les HMSC vérifiant le second point sont appelés des HMSC réguliers.

Par exemple, considérons le MSC c décrit dans la figure 1.5. Le HMSC $c^\star$ n'est pas régulier car le graphe de communication de c n'est pas fortement connexe. Cependant, si on considère $c_{p,q}$, la restriction de c aux processus p et q, alors, le HMSC $c^\star_{p,q}$ est un HMSC régulier.

Les HMSC réguliers ne sont cependant pas assez expressifs. En effet, il n'existe aucun HMSC régulier qui permet d'exprimer le fait qu'un processus peut envoyer autant de messages qu'il veut à un autre avant de recevoir une réponse. En effet, le langage de linéarisation engendré par une telle spécification correspond à un langage L où, pour chaque mot $u \in L$, chaque préfixe de u contient au moins autant d'émissions de messages que de réceptions, ce qui n'est pas un langage régulier.

Reconnaissabilité

Une autre série de travaux concerne la caractérisation de sous-ensembles reconnaissables de $\mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$. De tels sous-ensembles n'ont pas nécessairement un langage de linéarisation régulier et ne peuvent donc pas être simulés par des automates. Cependant, lorsque l'on considère comme élément de base de l'exécution, non plus des événements isolés, mais des morceaux d'ordres indivisibles appelés *atomes*, il est possible de construire un modèle opérationnel séquentiel qui simule le comportement de telles spécifications.

Afin de définir une notion de reconnaissabilité pour les HMSC qui soit plus générale que celle énoncée précédemment sur le langage de linéarisation, il faut plonger $\mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$ dans le monoïde de traces. Pour cela, nous associons naturellement à $\mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$ une relation d'indépendance I_{loc} définie de la manière suivante : pour tout MSC m et m', $(m, m') \in I_{loc}$ si $\{loc(e) \mid e \in E_m\} \cap \{loc(e) \mid e \in E_{m'}\} = \emptyset$, c'est-à-dire si les deux MSC ne partagent pas de localités. Clairement, $(m, m') \in I_{loc}$ implique que $m \circ m' = m' \circ m$. Un MSC m non vide est appelé un *atome* ([Alur 99, Hélouët 00b]), ou premier ([Morin 02]), si $m = m_1 \circ m_2$ implique que $m_1 = \varepsilon$ où $m_2 = \varepsilon$.

Hélouët et le Maigat ont montré dans [Hélouët 00b] que tout élément de $\mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$ peut se décomposer en un produit d'atomes. De plus, Morin montre dans [Morin 02] que cette décomposition est unique, modulo commutations compatibles avec la relation d'indépendance définie sur les atomes. Plus précisément :

Théorème 1.23 ([Morin 02]) *Soit* Γ*, un sous-ensemble d'atomes de* $\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})$*. Alors, le morphisme* $\eta : \mathbb{M}(\Gamma, I_{loc}) \to \langle\Gamma\rangle^{\star}_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})}$*, qui associe à chaque trace* $[a_1 \dots a_n]_{\sim_{I_{loc}}}$ *un* MSC $a_1 \circ \dots \circ a_n$*, est un isomorphisme.*

Par conséquent, les résultats applicables sur les traces sont applicables sur $\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})$. Il suffit, en effet, de considérer les noms des atomes plutôt que les morceaux d'ordres qu'ils représentent. Nous notons, donc $\Gamma^{\star}$ l'ensemble des MSC produits par concaténation des éléments de Γ et $\langle\Gamma\rangle^{\star}$ le monoïde libre engendré par concaténation des noms d'atomes, et nous étendons naturellement I_{loc} aux noms d'atomes. Ainsi, étant donné un ensemble d'atomes Γ, le théorème précédent permet d'identifier la classe des langages de traces reconnaissables aux ensembles de MSC dont le langage d'atomes est le langage d'un $\langle\Gamma\rangle^{\star}$-automate. Par conséquent, savoir, pour un HMSC quelconque, s'il existe un automate d'atomes qui a le même langage d'atomes est décidable, si, et seulement si, la fermeture réflexive de I_{loc} est transitive. Donc, en général, savoir si le langage d'un HMSC est reconnaissable est indécidable.

D'autre part, la reconnaissablité d'un langage de MSC peut être décidée pour une sous-classe de HMSC appelés HMSC globalement coopératifs. De plus, cette restriction est testable par un algorithme CoNP-complet.

Définition 1.24 (globalement coopératifs) *Soient* Γ*, un ensemble d'atomes et* α*, un* HMSC *tel que* $\mathcal{L}_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})}(\alpha) \subseteq \langle\Gamma\rangle^{\star}_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})}$*. Alors,* α *est* globalement coopératif *si, pour toute sous-expression* $\beta^{\star}$ *de* α*, la décomposition en atomes de* β *est corationnelle dans* $\mathbb{M}(\Gamma, I_{loc})$*, c'est-à-dire si le graphe de communication de tout* MSC *appartenant à* $\mathcal{L}_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})}(\beta)$ *est D-connexe.*

Nous pouvons maintenant introduire le résultat principal de cette partie, qui permet d'identifier les langages reconnaissables de $\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})$. Il suffit, en effet, d'utiliser l'isomorphisme η du théorème 1.23, afin d'identifier l'image inverse des reconnaissables du monoïde de traces $\mathbb{M}(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, I_{loc})$, caractérisés par le théorème 1.12 :

Théorème 1.25 ([Morin 02]) *Soient* Γ*, un ensemble d'atomes,* I_{loc}*, la relation d'indépendance sur ces atomes,* $L \subseteq \Gamma^{\star} \subseteq \mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})$*, un langage de* MSC *et* $\eta : \mathbb{M}(\Gamma, I_{loc}) \to \langle\Gamma\rangle_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})}$*, l'isomorphisme du théorème 1.23. Alors, les affirmations suivantes sont équivalentes :*

- *L est reconnaissable dans* $\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})$ *;*
- $\eta^{-1}(L)$ *est reconnaissable dans le monoïde de traces* $\mathbb{M}(\Gamma, I_{loc})$ *;*
- $\bigcup \eta^{-1}(L)$ *est régulier dans* $\Gamma^{\star}$*, le monoïde libre engendré par* Γ *;*
- *il existe un* HMSC *globalement coopératif* α *tel que* $\mathcal{L}_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})} = L$ *;*
- *il existe une trace corationnelle* α *de* $REX(\mathbb{M}(\Gamma, I_{loc}))$ *telle que* $\mathcal{L}_{\mathbb{M}(\Gamma,I_{loc})}(\alpha) = \eta^{-1}(L)$ *;*
- *il existe une trace corationnelle* α *de* $REX(\Gamma^{\star})$ *telle que* $[\mathcal{L}_{\Gamma^{\star}}(\alpha)]_{\sim_{I_{loc}}} = \bigcup \eta^{-1}(L)$*.*

La figure 1.6 montre le $\{a,b\}^{\star}$-automate $\mathcal{A}_{\alpha}$ associé au HMSC $\alpha = ab^{\star}a$, avec a et b dans $\mathbb{MSC}(\{p,q\},\{m,n\},\emptyset)$. La figure de droite montre un MSC qui peut être généré par cet automate, correspondant au chemin $q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_2 \xrightarrow{a} q_3$ dans $\mathcal{A}_{\alpha}$. α est un HMSC globalement coopératif. Par conséquent, son langage d'atomes est régulier dans le monoïde libre $\Gamma(\alpha)^{\star}$, mais son langage de linéarisation n'est pas régulier dans $\Sigma^{\star}_{\mathcal{P},\mathcal{M},\mathcal{I}}$ car il n'existe pas de HMSC

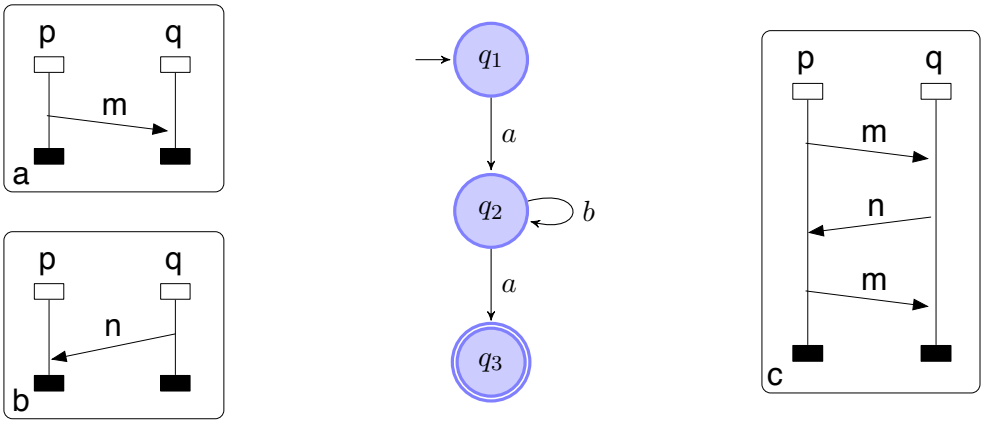


FIG. *1.6 – A gauche, deux* MSC a *et* b *qui sont dans* $\mathbb{MSC}(\{p,q\},\{m,n\},\emptyset)$. *Au centre un* $\{a,b\}^\star$*-automate* $\mathcal{A}$ *qui reconnait le* HMSC $ab^\star a$. *A droite,* $c \in \mathcal{L}_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})}(\mathcal{A}) \subseteq \mathbb{MSC}(\{p,q\},\{m,n\},\emptyset)$, *car* $c = a \circ b \circ a$.

régulier ayant le même langage.

Pour résumer, nous avons identifié deux classes principales des HMSC. Une première, celle des réguliers, correspond aux modèles de spécification de systèmes communicants dont le langage des entrelacements d'exécutions peut être simulé par des modèles opérationnels séquentiels. Une seconde, celle des globalement coopératifs, correspond aux modèles de spécification de systèmes communicants dont le modèle d'exécution est un modèle global de "haut niveau", équivalent à l'enchaînement régulier de phases d'un protocole. Ces phases peuvent, cependant, se dérouler parallèlement si elles ne partagent pas les mêmes processus. Dans les deux cas, le comportement obtenu est finiment engendré car les scénarios générés par un modèle sont engendrés par un nombre fini de MSC.

Cependant, lorsque l'on cherche à spécifier certains comportements de systèmes parallèles et répartis, il est intéressant de pouvoir décrire des systèmes qui ne sont pas finiment engendrés. Pour cela, nous introduisons, dans la partie suivante, une extension des MSC qui permet de décrire des comportements de base non clos par communication (c'est-à-dire des comportements dans lesquels l'on s'autorise à couper des messages, contrairement aux MSC).

1.3.3 HMSC contextuels

Dans la partie précédente, nous avons présenté un modèle normalisé de spécification de systèmes communicants. Nous présentons, dans cette partie, une extension qui en étend considérablement l'expressivité, puisque ce modèle permet d'exprimer des langages de MSC qui ne sont pas finiment engendrés. Celui-ci, appelé HMSC compositionnel, a été introduit par Gunter, Muscholl et Peled dans [Gunter 01], où il a été montré que de nombreux problèmes, décidables pour les HMSC, devenaient indécidables dans ce nouveau cadre. Ensuite, Genest, Kuske et Muscholl ont identifié, dans [Genest 06a], une sous-classe syntaxique de ce langage de spécification suffisamment expressive pour décrire des comportements non finiment engendrés, tout en gardant des propriétés décidables de vérification et de supervision.

Cependant, dans ces travaux, les modèles ne sont pas donnés sous forme d'ensembles par-

ticuliers d'un monoïde : dans [Gunter 01] l'opérateur de composition n'est pas associatif et dans [Genest 06a], ce n'est pas une loi interne. Afin de conserver les notations utilisées depuis le début de ce chapitre, nous introduisons plutôt un autre modèle, les HMSC contextuels, défini par Baudru et Morin dans [Baudru 07] et qui se fonde sur la notion de monoïde. Le pouvoir d'expression, en terme de famille de MSC engendrés, des HMSC contextuels est équivalent à celui des HMSC compositionnels sûrs, HMSC compositionnels dont chaque exécution génère exactement un MSC (il n'y a donc pas plusieurs manières de recoller les messages). Ainsi, dans la suite, CHMSC fera référence, indifféremment, aux HMSC contextuels ou aux HMSC compositionnels sûrs.

Tout d'abord, comme nous voulons nous autoriser à couper les messages dans les MSC servant de base aux expressions rationnelles, nous associons un contexte de messages à chaque MSC. Grâce à ce contexte, nous redéfinissons $\ll$, la relation qui apparie les envois et les réceptions de messages, pour une paire (o, χ), avec $o \in \mathbb{P}(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D_{loc})$ un pomset étiqueté par le même alphabet qu'un MSC, et $\chi \in \mathbb{N}^{\mathcal{P}\times\mathcal{M}\times\mathcal{P}}$ une fonction qui à chaque canal de communication donne le nombre de messages en attente. Etant donné une paire (o, χ), $\ll \subseteq (E_p \times E_p)$ est défini de la manière suivante : $e \ll e'$ si il existe $p, q \in \mathcal{P}$ et $m \in \mathcal{M}$ tels que $e = p!q(m)$, $e' = q?p(m)$ et le nombre d'envois et de réceptions de messages de type m dans le passé causal des deux événements est le même, c'est-à-dire $\chi(p, m, q) + |I_o(e)|_{p!q(m)} = |I_o(e')|_{q?p(m)}$. Par définition, la politique de routage des messages est donc weak-FIFO, comme pour les MSC. Nous étendons maintenant la définition 1.20 d'un MSC afin de permettre à certains messages de n'être pas reçus.

Définition 1.26 (cMSC) *Un* MSC contextuel *est une paire (c, χ) où $c \in \mathbb{P}(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D_{loc})$ est un pomset et $\chi \in \mathbb{N}^{\mathcal{P}\times\mathcal{M}\times\mathcal{P}}$ est un état de canal de communication tel que :*

(i) tous les événements étant sur la même localité sont totalement ordonnés, c'est-à-dire que, $\forall e, e' \in E$ tels que $loc(e) = loc(e')$, $e \leq e'$ ou $e' \leq e$;

(ii) *(c,χ) a* envoyé plus de messages d'un même type qu'il n'en a reçus*, c'est-à-dire que pour tout $p, q \in \mathcal{P}$ et $m \in \mathcal{M}$, $\chi(p, m, q) + |c|_{p!q(m)} \geq |c|_{q?p(m)}$;*

(iii) c ne rajoute pas d'autre ordre que la causalité locale et induite par l'échange de messages, c'est-à-dire que la relation $\leq$ est exactement $(\ll \bigcup_{p\in\mathcal{P}} (\leq \cap E_p^2)^*$.

Seul le point (ii) est modifié par rapport à la définition d'un MSC*. Nous notons $c\mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$ l'ensemble des* MSC *contextuels.*

De la même manière que dans les parties précédentes, nous munissons l'ensemble des MSC contextuels d'une loi de composition interne pour obtenir un monoïde. Cette loi va, comme pour la loi de composition des MSC, recoller localement les exécutions de chaque processus, et faire, en plus, que les messages envoyés en attente soient correctement réceptionnés. Avant de définir plus formellement cette loi de composition, nous introduisons un élément particulier, nommé 0 et appelé élément *non-valide*, et qui est un élément absorbant. En effet, nous fixons $x \circ 0 = 0 \circ x = 0$. De même, nous introduisons un CMSC neutre noté ε, tel que pour tout x, nous fixons $x \circ \varepsilon = \varepsilon \circ x = x$.

Définition 1.27 (composition de cMSC) *Soient (c_1, χ_1) et (c_2, χ_2), deux* MSC *contextuels, et $\rightsquigarrow \subseteq E_{c_1} \times E_{c_2}$, la relation telle que : $e_1 \rightsquigarrow e_2$ si $\lambda_{c_1} = p!q(m)$, $\lambda_{c_2} = q?p(m)$ et que le nombre d'envois de m précédant e_1 est le même que le nombre de réceptions de m précédant e_2, c'est-à-dire $\chi_1(p, m, q) + |I_{c_1}(e_1)|_{p!q(m)} = |c_1|_{q?p(m)} + |I_{c_2}(e_2)|_{q?p(m)}$.*

Tout d'abord, si pour tout $p, q \in \mathcal{P}$ *et* $m \in \mathcal{M}$*, le nombre de messages en attente dans* (c_2, χ_2) *est exactement le nombre de messages envoyés dans* (c_1, χ_1)*, c'est-à-dire si* $\chi_1(p,m,q) + |c_1|_{p!q(m)} - |c_1|_{q?p(m)} = \chi_2(p,m,q)$*, alors* $(c_1, \chi_1) \circ (c_2, \chi_2) = (c_3, \chi_3)$*, où* $c_3 = [E, \leq, \lambda]$*, avec :*

- $E = E_{c_1} \uplus E_{c_2}$ *;*
- $\leq$ *est la fermeture transitive de* $\leq_{c_1} \cup \leq_{c_2} \cup \{(e_1, e_2) \in E_{c_1} \times E_{c_2} \mid loc(e_1) = loc(e_2)\} \cup \rightsquigarrow$ *;*
- $\lambda = \lambda_{c_1} \cup \lambda_{c_2}$ *;*
- $\chi_3 = \chi_2$.

Ensuite, si $(c_1, \chi_1) = \varepsilon$ *alors* $(c_1, \chi_1) \circ (c_2, \chi_2) = (c_2, \chi_2)$ *(et inversement si* $(c_2, \chi_2) = \varepsilon$*). Enfin, dans les autres cas,* $(c_1, \chi_1) \circ (c_2, \chi_2) = 0$.

Proposition 1.28 ([Baudru 07]) $(c\mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I}), \circ, \varepsilon)$ *est un monoïde.*

Ensuite, de la même façon que nous l'avons fait pour les MSC dans la partie précédente, nous introduisons le langage de spécifications rationnelles permettant d'exprimer des familles infinies de MSC.

Définition 1.29 (cHMSC) *Un* cHMSC *est une expression rationnelle de* $c\mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$.

Classiquement, le langage d'un cHMSC α est l'ensemble des cMSC généré par l'expression rationnelle α. Cependant, dans la suite de ce document, nous nous intéressons uniquement au langage de MSC généré par α, que nous notons abusivement $\mathcal{L}_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})}(\alpha)$. $\mathcal{L}_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})}(\alpha)$ est donc l'ensemble des MSC m (donc des cMSC clos par communication) tels que (m, χ) soit généré par α (ce que l'on pourrait écrire, $(m, \chi) \in \mathcal{L}_{c\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})}(\alpha)$). La définition des classes syntaxiques régulières et globalement coopératives définies sur les HMSC s'étendent sans aucun problème aux cHMSC. Cependant, ces nouvelles classes possèdent des propriétés différentes. En particulier, le théorème 1.22 peut se généraliser aux langages de MSC qui ne sont pas finiment engendrés afin d'étendre l'équivalence de Kleene au monoïde des MSC : pour les systèmes communicants, les spécifications exprimées à l'aide de cHMSC réguliers (c'est-à-dire ceux dont tous les MSC, générés par des sous-expressions qui peuvent s'itérer, ont un graphe de communication fortement connexe) ont le même pouvoir d'expression que les modèles opérationnels séquentiels qui génèrent des linéarisations de MSC. Ce théorème a, à l'origine, été démontré pour les HMSC compositionnels sûrs et réguliers que l'on peut facilement montrer être exactement les HMSC contextuels réguliers.

Théorème 1.30 ([Genest 06a]) *Soit* $L \subseteq \mathbb{MSC}(\mathcal{M}, \mathcal{P}, \mathcal{I})$*. Les affirmations suivantes sont équivalentes :*

- $Lin(L)$ *est régulier ;*
- *il existe un* cHMSC α *régulier tel que* $\mathcal{L}_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})}(\alpha) = L$.

La figure 1.7 donne un exemple de cHMSC régulier, correspondant à l'expression rationnelle $ab^\star c$ (à gauche) et un MSC engendré par ce modèle (à droite). Le langage de linéarisations de ce cHMSC correspond au langage de l'expression rationnelle $p!q(m)\big(q!p(n)p?q(n)p!q(o)q?p(o)\big)^\star q?p(m)$. Le contenu des canaux de communication (χ) est indiqué à l'intérieur des boîtes englobantes des cMSC.

De plus, nous verrons, dans la partie suivante, que les cHMSC globalement coopératifs correspondent eux aussi à un modèle d'exécution spécifique, bien plus expressif que le modèle

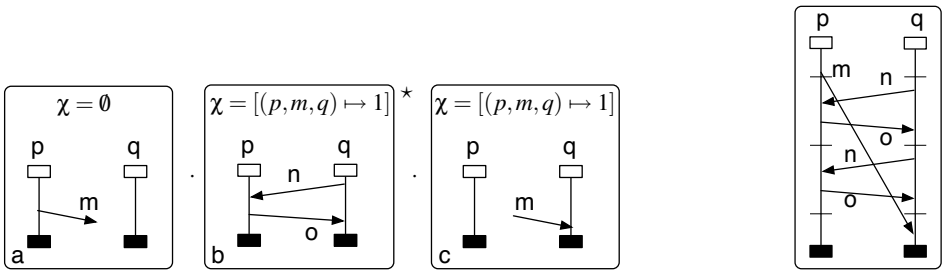


FIG. *1.7 – Un* cHMSC *qui génère un langage de* MSC *non finiment engendré.*

opérationnel séquentiel. Ce modèle d'exécution correspond à un ensemble de processus séquentiels évoluant de manière indépendante et communiquant par échange asynchrone de messages.

1.3.4 Réseaux d'automates communicants

Dans les parties précédentes, nous avons présenté plusieurs langages de spécification pour les systèmes communicants. Nous avons essayé de caractériser les cas où de tels modèles de haut niveau avaient le même pouvoir d'expression qu'un modèle de plus bas niveau, représenté par un automate. Nous avons vu, que, dans la plupart des cas, les langages de spécification définis étaient beaucoup plus expressifs que ce modèle d'exécution séquentiel et qu'il était difficile d'exprimer des spécifications complexes avec ces langages limités. En particulier, il n'est pas possible d'exprimer le fait qu'un processus envoie un nombre indéterminé de messages à un autre. Il est donc important de présenter un modèle d'exécution adapté, et c'est ce que nous faisons dans cette partie, en présentant le modèle des automates communicants.

Les réseaux d'automates communicants, appelé "Communicating Finite State Machines" (CFM) dans [Brand 83] ou "Message Passing Automata" (MPA) dans [Mukund 00], sont des modèles opérationnels simples qui permettent de spécifier et de décrire des protocoles. Ce modèle consiste en un ensemble fini de systèmes séquentiels à événements discrets qui communiquent par échange asynchrone de messages, par l'intermédiaire de canaux de communication sûrs et qui ne sont pas forcément bornés. Le travail de spécification d'un protocole est donc grandement facilité, car la politique de routage des messages est prise en compte dans le modèle lui-même. Ainsi, la sémantique formelle de langages normalisés de spécification, comme SDL ou Estelle, a été donnée à l'aide de réseaux d'automates communicants (se reporter par exemple à [Turner 93] pour plus de précisions). Cependant, les réseaux d'automates communicants ont le pouvoir d'expression des machines de Turing. Par conséquent, de nombreux problèmes sont indécidables pour ce modèle.

Avant de commencer à énoncer les définitions et principaux résultats associés aux réseaux d'automates communicants, fixons la même architecture que pour les MSC, à savoir, un ensemble de processus $\mathcal{P}$, un ensemble de messages $\mathcal{M}$, et un ensemble d'actions internes $\mathcal{I}$, à partir desquels nous pouvons définir l'ensemble des canaux de communication $Ch = \mathcal{P} \times \mathcal{M} \times \mathcal{P}$, et l'alphabet global $\Sigma_{\mathcal{M},\mathcal{P},\mathcal{I}}$ dont les lettres sont du type $p(a)$, $p!q(m)$ et $p?q(m)$, avec $p, q \in \mathcal{P}$, $a \in \mathcal{I}$ et $m \in \mathcal{M}$. Nous donnons maintenant la définition d'un réseau d'automates communicants. Cette définition est très proche de celle donnée pour les réseaux

d'automates asynchrones (déf. 1.13). En effet, les structures sont identiques. C'est seulement la construction de l'automate global (qui peut être infini dans le cas des réseaux d'automates communicants) qui est différente. En effet, la construction de l'automate global dépend de la sémantique des communications du modèle choisi : pour les automates asynchrones, c'est par mémoire partagée (les automates locaux peuvent lire un certain nombre de variables partagées) et pour les automates communicants, c'est par envois et réceptions de messages : il faut donc gérer des files de communication qui ne sont pas forcément bornées.

Définition 1.31 (automates communicants) *Un* réseau de $\Sigma^{\star}_{\mathcal{P},\mathcal{M},\mathcal{I}}$-automates *est une structure* $\mathcal{A} = \langle S_i, (\mathcal{A}_p)_{p\in\mathcal{P}}, S_f\rangle$ *où, pour tout* $p \in \mathcal{P}$, $\mathcal{A}_p$ *est un* $\Sigma^{\star}_p$*-automate* $(S^p, \rightarrow_p, \Sigma_p, S^p_i, S^p_f)$, *telle que :*

- S_i *est un ensemble d'états initiaux globaux associés à une file vide, c'est-à-dire* $S_i \subseteq S_i^1 \times \ldots \times S_i^{|\mathcal{P}|}$;
- S_f *est un ensemble d'états finaux globaux associés à une file vide, c'est-à-dire* $S_f \subseteq S_f^1 \times \ldots \times S_f^{|\mathcal{P}|}$.

Un état global $g \in G$ *est un couple* (χ, s), *où* $\chi : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{M}^{\star}$ *est l'état de la file de communication entre deux processus (*$\chi(p,q)$ *représentant les messages en attente d'être reçus entre* p *et* q*) et* $s \in S$ *est est un* $|\mathcal{P}|$*-uplet d'états locaux, c'est-à-dire* $S = S_i^1 \times \ldots \times S_i^{|\mathcal{P}|}$. *La* relation de transition globale, *notée* $\rightarrow \subseteq G \times \Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}} \times G$ *est définie comme suit :*

- $(\chi, (s_1, \ldots, s_{|\mathcal{P}|})) \xrightarrow{p!q(m)} (\chi', (s'_1, \ldots, s'_{|\mathcal{P}|}))$ *si*
  - $s_p \xrightarrow{p!q(m)}_p s'_p$ *et* $s_t = s'_t$ *pour* $t \neq p$;
  - $\chi'(p,q) = \chi(p,q) \cdot m$ *et* $\chi(r,s) = \chi'(r,s)$ *pour* $r \neq p, q \neq s$.
- $(\chi, (s_1, \ldots, s_{|\mathcal{P}|})) \xrightarrow{p?q(m)} (\chi', (s'_1, \ldots, s'_{|\mathcal{P}|}))$ *si*
  - $s_p \xrightarrow{p?q(m)}_p s'_p$ *et* $s_t = s'_t$ *pour* $t \neq p$;
  - $\chi(p,q) = m \cdot \chi'(p,q)$ *et* $\chi(r,s) = \chi'(r,s)$ *pour* $r \neq p, q \neq s$.

L'automate infini global est noté $\langle S_i, \|_{p\in\mathcal{P}}\ \mathcal{A}_p, S_f\rangle$ *(*$\|$ *dénotant la mise en parallèle d'automates à l'aide de files de communication FIFO).*

Remarquons tout de suite que la politique de routage des automates communicants est différente de celle utilisée pour les MSC ou les CMSC. En effet, dans ce cas, deux messages ayant des types différents mais utilisant un même canal ne peuvent se croiser. Cette politique de routage est appelée FIFO. Ainsi, il est clair que de tels modèles opérationnels ne peuvent simuler qu'une sous-classe des MSC (qui sont eux définis avec une politique de routage weak-FIFO).

Ensuite, de la même façon que pour les réseaux d'automates asynchrones, un réseau d'automates communicants est dit *déterministe* si $\rightarrow$ est une fonction de $G \times \Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$ dans G. Remarquons que cette définition est moins restrictive que celle de [Henriksen 05].

De même, selon que le réseau est à états initiaux locaux (c'est-à-dire si S_i est le produit cartésien des états initiaux locaux), à états finaux locaux (c'est-à-dire si S_f est le produit cartésien des états finaux locaux), ou à états initiaux et finaux locaux, nous notons l'automate infini global respectivement $\langle \|_{p\in\mathcal{P}}\ \mathcal{A}_p, S_f\rangle$, $\langle S_i, \|_{p\in\mathcal{P}}\ \mathcal{A}_p\rangle$ ou $\|_{p\in\mathcal{P}}\ \mathcal{A}_p$, car les ensembles qui ne sont pas notés peuvent être aisément reconstruits. Enfin, un tel réseau est dit sans blocage si tous les états globaux accessibles sont co-accessibles. Le langage d'un réseau d'automates communicants $\mathcal{A} = \langle S_i, (\mathcal{A}_p)_{p\in\mathcal{P}}, S_f\rangle$, noté $\mathcal{L}_{\Sigma^{\star}_{\mathcal{P},\mathcal{M},\mathcal{I}}}(\langle S_i, \|_{p\in\mathcal{P}}\ \mathcal{A}_p, S_f\rangle$ est l'ensemble des mots

$u = a_1 \dots a_{|u|}$ tel qu'il existe un chemin $S_0 \xrightarrow{a_1} \dots \xrightarrow{a_{|u|}} S_{|u|}$ acceptant dans $\langle S_i, \|_{p\in\mathcal{P}}\, \mathcal{A}_p, S_f\rangle$.

Etant donné un réseau $\mathcal{A}$ d'automates communicants, nous disons que $\mathcal{A}$ est *(universellement) borné* si il existe un entier b tel que tous les états globaux accessibles possèdent des canaux de communication contenant moins de b messages. Rappelons ensuite que, grâce à la politique de routage FIFO mise en œuvre dans la définition d'un réseau d'automates communicants, pour chaque mot u de $\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$ généré par $\mathcal{A}$ et possédant le même nombre d'envois et de réceptions pour chaque type de message, nous pouvons construire un unique MSC FIFO m_u ayant u comme linéarisation. Nous dirons, ainsi, que $\mathcal{A}$ est *existentiellement borné* si, pour tout $u \in \mathcal{L}_{\Sigma^\star_{\mathcal{P},\mathcal{M},\mathcal{I}}}(\mathcal{A})$, il existe $v \in \mathcal{L}_{\Sigma^\star_{\mathcal{P},\mathcal{M},\mathcal{I}}}(\mathcal{A})$ tel que m_u soit isomorphe à m_v et que tous les états globaux atteints pour générer v possèdent des canaux de communication contenant moins de b messages.

Genest, Kuske et Muscholl ont étendu, dans [Genest 06a], les résultats donnés par Enriksen et al. dans [Henriksen 05] sur l'équivalence entre spécifications rationnelles, données en terme de cHMSC réguliers ou globalement coopératifs, et modèles d'exécution, donnés en terme de réseaux d'automates communicants. De même, Baudru et Morin ont caractérisé plus finement, dans [Baudru 07], la classe des réseaux qui peuvent mettre en œuvre des cHMSC réguliers.

Théorème 1.32 *Soit $L \subseteq \mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$, un langage de* MSC *FIFO. Les deux affirmations suivantes sont équivalentes [Baudru 07] :*

- *L est le langage d'un* cHMSC *régulier ;*
- *L est le langage (avec ajout de données) d'un réseau universellement borné et sans blocage d'automates communicants à états finaux locaux.*

De même, les deux affirmations suivantes sont équivalentes [Genest 06a] :

- *L est le langage d'un* cHMSC *globalement coopératif ;*
- *L est le langage (avec ajout de données) d'un réseau d'automates communicants existentiellement borné.*

Pour plus de précisions sur les liens entre les différentes classes des réseaux d'automates communicants et ensembles rationnels de $\mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$, se rapporter à l'état de l'art du chapitre 2 (partie 2.2.1), consacré à la mise en œuvre des cHMSC par des réseaux d'automates communicants. Notons que la preuve des résultats présentés ici s'appuie sur le théorème de Zielonka (th. 1.14). Les algorithmes associés ont donc un prix élevé : la traduction d'un modèle vers un autre est toujours, au moins, exponentielle.

Finalement, nous concluons ce chapitre en donnant les résultats de complexité pour les différents problèmes de décision évoqués précédemment (appartenance, inclusion et vacuité de l'intersection). Sans surprise, ces problèmes sont difficiles dans le cadre des systèmes communicants car les canaux de communication ne sont pas forcément bornés.

Proposition 1.33 *Pour les modèles de haut niveau de systèmes communicants, nous avons les résultats de complexité suivants, triés verticalement par problème et horizontalement par générateur des langages L et L′ (avec gc pour globalement coopératif et reg pour régulier) :*

	$REX(\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I}))$	L *est gc-(c)*HMSC	L *est reg-(c)*HMSC
$x \in L$	*NP-c[Alur 01]*	*NP-c[Alur 01]*	*NP-c[Alur 01]*
$L \cap L' = \emptyset$	*Indécidable*	*PSPACE-c[Genest 06a]*	*PSPACE-c[Alur 99]*
$L' \subseteq L$	*Indécidable*	*EXPSPACE-c[Genest 06a]*	*EXPSPACE-c[Alur 99]*

Pour les modèles de bas niveau de systèmes communicants, nous avons les résultats de complexité suivants (avec RAC pour réseau d'automates communicants) :

	RAC	*RAC borné*
$x \in L$	*NP*	*P*
$L \cap L' = \emptyset$	*Indécidable*	*PSPACE*
$L' \subseteq L$	*Indécidable*	*EXPSPACE*

La proposition précédente montre que les différents problèmes ont des complexités comparables, quelque soit le niveau où l'on se place. Cependant, les modèles de haut niveau possèdent une concision plus grande, de l'ordre d'un facteur exponentiel. Au final, il est donc beaucoup plus efficace de résoudre ces problèmes directement à partir des spécifications.

1.4 Des modèles mixtes ?

Dans ce chapitre, nous avons vu deux types de modèle parallèle, dédiés à la spécification de systèmes asynchrones d'une part, et à la spécification de systèmes communicants, d'autre part. Ces différents modèles sont décrits à l'aide d'expressions rationnelles et ont une interprétation sémantique directe en tant qu'ensembles rationnels d'un monoïde adapté : pour les systèmes asynchrones, celui des traces de Mazurkiewicz et pour les systèmes communicants, celui des MSC. Pour ces deux types de systèmes, nous avons aussi vu les modèles de bas niveau correspondants : les réseaux d'automates asynchrones et les réseaux d'automates communicants.

Dans la suite de ce document, nous généralisons ces deux méthodes pour proposer des modèles mixtes. Les modèles de haut niveau seront toujours des expressions rationnelles, mais le monoïde sous-jacent sera celui des pomsets. En effet, les pomsets généralisent parfaitement les deux types d'approche, ce qui nous permettra d'adapter un grand nombre de techniques venues du domaine des systèmes parallèles asynchrones et des systèmes parallèles communicants. De plus, dans le chapitre suivant, nous proposons un modèle de bas niveau adapté pour ce type de systèmes mixtes : des réseaux d'automates localement asynchrones et globalement communicants.

Chapitre **2**

Modèles mixtes

Ce chapitre constitue, avec le chapitre précédent, la partie I de cette thèse, consacrée aux *modèles parallèles*. Plus précisément, nous avons présenté, dans le chapitre précédent, deux modèles classiques pour spécifier des systèmes parallèles et répartis. Le premier de ces modèles permet de modéliser les systèmes asynchrones, à l'aide d'expressions rationnelles du monoïde de traces. Le second de ces modèles permet de modéliser les systèmes communicants, à l'aide d'expressions rationnelles du monoïde des MSC.

Dans ce chapitre, nous généralisons les deux approches précédentes, afin de proposer un modèle d'interactions mixtes, c'est-à-dire localement asynchrones et globalement communicantes. Nous introduisons, pour cela, le modèle des HMSC causaux, un modèle de haut niveau qui étend le formalisme des HMSC en permettant des commutations d'événements comme pour les traces de Mazurkiewicz.

Plus précisément, nous nous intéressons, dans ce chapitre, à deux problèmes concernant des systèmes parallèles et répartis *bornés*.

Le premier problème concerne la mise en œuvre de HMSC causaux par des automates et des réseaux bornés d'automates communicants. Pour cela, nous définissons une classe syntaxique, appelée HMSC causaux réguliers, dont tous les membres sont équivalents à des automates et à des réseaux bornés d'automates communicants.

Le second problème concerne l'équivalence entre une classe syntaxique des HMSC causaux dits faiblement réguliers et cohérents et un modèle de bas niveau appelé réseaux bornés d'automates mixtes, car localement asynchrones et globalement communicants.

Enfin, les parties II et III de ce document seront consacrées, respectivement, à la vérification et à la supervision de ces systèmes parallèles et répartis mixtes.

Contexte

Lorsque l'on cherche à modéliser et analyser un système parallèle et réparti dont les entités sont limitées à un seul type d'interaction, nous avons vu, dans le chapitre précédent, que nous disposons de nombreux outils. Les modèles de haut niveau sont définis et largement étudiés, qu'il s'agisse d'expressions rationnelles de traces ou de MSC. De même, de nombreux travaux décrivent le lien avec les modèles de bas niveau correspondants, c'est-à-dire les réseaux d'automates asynchrones et les réseaux d'automates communicants. De plus, pour tous ces modèles, des techniques efficaces d'analyses existent, fondées sur une complexité raisonnable pour les problèmes de l'appartenance, de l'inclusion et de la vacuité de l'intersection des langages considérés.

Cependant, lorsque l'on s'intéresse à des systèmes dont les entités ont des modes d'interactions mixtes (qui dépendent, par exemple, de leur répartition), le panorama est plus disparate et les techniques d'analyse automatique sont rares.

Tout d'abord, citons les réseaux de Petri ([Reutenauer 90]), qui peuvent constituer un bon modèle de bas niveau. Cependant, ces réseaux imposent une politique de routage unique (en l'occurence weak-FIFO) dans le cas non-borné. Dans le cas borné, la politique FIFO peut être simulée, mais la complexité de la traduction est grande. En outre, la complexité des problèmes de décision associés à ces modèles rend toute mise en œuvre irréalisable en pratique. Ainsi, le problème de l'atteignabilité pour les réseaux de Petri a une complexité qui est entre EXPSPACE et non élémentaire (voir [Esparza 94] pour un résumé des principaux résultats concernant les réseaux de Petri).

Ensuite, citons une version légèrement étendue des automates cellulaires ([Bollig 05]), qui a le même pouvoir d'expression que le fragment existentiel de la logique monadique du second ordre. Cette approche permet d'unifier les réseaux d'automates asynchrones et les réseaux d'automates communicants. Cependant, cette unification se base sur la logique monadique du second ordre, ce qui rend, là encore, les problèmes de décision qui nous intéressent irréalisables en pratique.

Enfin, les pomsets sont un modèle de haut niveau très étudié depuis deux décennies ([Gischer 88, Pratt 86]). Esik et Okawa ont ainsi montré, dans [Ésik 99], que les langages rationnels de pomsets (avec deux opérateurs de composition séquentielle et parallèle qui correspondent, respectivement, à $D = \Sigma^\star$ et $D = \emptyset$ dans notre formalisme) ne peuvent pas être décrits à l'aide d'un nombre fini d'équations. Pour pallier ce problème, d'autres travaux se sont intéressés à représenter des ensembles infinis de pomsets, de manière finie. Citons par exemple les travaux de Fanchon et Morin ([Fanchon 02]) qui ont cherché à caractériser des ensembles reconnaissables de pomsets. Cependant, dans tous ces travaux, la théorie dont nous avons besoin pour comparer différents modèles définis à l'aide de rationnels de pomsets n'est pas disponibles, ou alors avec des coûts trop élevés pour être réalisables en pratique.

Pour nos travaux, nous avons donc choisi de nous placer dans le cadre du monoïde des pomsets avec, à la manière du monoïde des traces, une loi de composition paramétrée par une relation de dépendance. Ainsi, ce modèle étend directement celui des HMSC et des traces de Mazurkiewecz, ce qui va nous permettre d'utiliser un grand nombre d'outils théoriques présents dans ces deux formalismes. En particulier, nous allons être capables de mettre au point des techniques efficaces d'analyse, pour des sous-classes bien particulières, inspirées des

sous-classes des HMSC et des traces, identifiées dans le chapitre précédent.

Plus précisément, dans ce chapitre, nous allons définir un nouveau modèle de haut niveau permettant de décrire des interactions mixtes, appelé HMSC causaux. Puis nous allons nous attacher à décrire le lien qui existe entre ce nouveau modèle de haut niveau et un mélange des modèles de bas niveau existants, que nous avons appelé réseau d'automates mixtes, car il étend le formalisme des réseaux d'automates asynchrones et des réseaux d'automates communicants.

Organisation du chapitre

Ce chapitre est organisé de la manière suivante. Tout d'abord, la partie 2.1 présente un nouveau modèle de haut niveau pour décrire des systèmes parallèles et répartis ayant des interactions mixtes, que nous avons appelé HMSC causaux. Ce modèle combine la concision des HMSC avec celle des traces de Mazurkiewicz. Ensuite, dans la partie 2.2, nous donnons un premier résultat de mise en œuvre avec ajout de données, d'une classe des HMSC causaux dits réguliers vers des réseaux d'automates communicants. Enfin, dans la partie 2.3, nous présentons un modèle de bas niveau pour les systèmes parallèles et répartis avec interactions mixtes. Puis, nous caractérisons la classe des HMSC causaux réguliers et cohérents qui correspond exactement (à renommage près) aux modèles bornés de ce type.

2.1 HMSC causaux

La nécessité d'inventer un nouveau modèle de haut niveau qui étend le formalisme des HMSC provient d'un double constat.

D'une part, les HMSC sont une représentation concise et intuitive des interactions entre entités d'un système communicant. Cependant, ils ne permettent d'exprimer que des comportements qui sont finiment engendrés, c'est-à-dire des comportements que l'on peut découper en un nombre fini de phases qui ne se chevauchent pas. La figure 2.1 illustre ce problème. Le MSC N de cette figure présente un scénario où un processus p envoie différentes requêtes Q au processus q, qui sont entrelacées avec les réponses A de q à p. Ce genre de comportement se retrouve par exemple dans le protocole du bit alterné (utilisé par les bus USB) ou le protocole de fenêtres glissantes (utilisé par TCP). En utilisant des HMSC classiques, uniquement des scénarios semblables au MSC M de la figure 2.1 peuvent être définis.

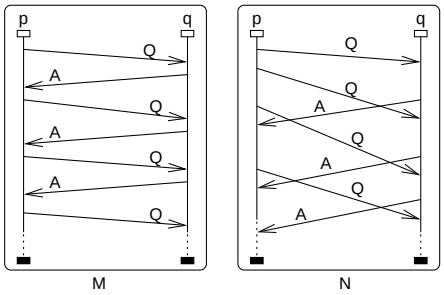


FIG. 2.1 – *A gauche, un* MSC *décomposable en plusieurs phases. A droite, un* MSC *non finiment engendré.*

D'autre part, certaines causalités imposées par un MSC ne sont pas intuitives, notamment celles liées à des "race conditions". Ces causalités apparaissent lorsque deux messages envoyés indépendamment doivent être reçus dans un ordre spécifié. Cet ordre dépend uniquement de la vitesse de transmission dans le réseau sous-jacent ; il est donc très difficile d'imposer cet ordre dans un scénario. Pour cela, Alur et al. dans [Alur 96], puis Muscholl et al. dans [Muscholl 98], font la distinction entre l'ordre visuel produit par un HMSC et l'ordre causal qui est une relaxation de cet ordre visuel dans le cas où la causalité est contre-intuitive.

Pour pallier les déficiences des HMSC, nous avons choisi, dans [Gazagnaire 07b], de *générer directement l'ordre causal*. Pour cela, nous proposons les deux changements suivants, dans la définition du monoïde des MSC pour obtenir celui des MSC causaux :

1. Au sein d'un MSC causal, nous relâchons la contrainte qui force chaque processus à être totalement ordonné, en autorisant n'importe quel ordre partiel au sein des MSC (avec la restriction que cet ordre permette de reconstruire la causalité liée aux messages) ;
2. Lors de la composition entre MSC causaux, nous relâchons la contrainte de la juxtaposition des phases, induite par la définition de la composition séquentielle faible en autorisant une loi de composition définie à partir de n'importe quelle relation de dépendance symétrique et réflexive.

Plus formellement, reprenons la définition des traces de la partie 1.2.1 et des MSC de la partie 1.3.2. En plus de ces notations, nous dirons qu'un alphabet concurrent $(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D)$ est *localisé* si, pour tout $a, b \in \Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$, $a\ D\ b$ implique que $loc(a) = loc(b)$. Intuitivement, cela veut dire que les synchronisations qui apparaissent lors de la composition n'interviennent qu'entre événements qui sont sur un même processus.

Définition 2.1 (MSC causal) *Un* MSC *causal c est un* MSC *pour lequel la contrainte de l'ordre total sur chaque processus a été enlevée. De plus, les événements qui correspondent à l'envoi (ou à la réception) d'un même message sont totalement ordonnés. Plus précisément, soit $(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D)$, un alphabet concurrent localisé. $c = [E, \leq, \lambda] \in \mathbb{P}(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D)$ est un* MSC *causal si :*

(i) *pour tous e et e', si $\lambda(e) = \lambda(e') = p!q(m)$ ou $\lambda(e) = \lambda(e') = p?q(m)$ pour $p, q \in \mathcal{P}$ et $m \in \mathcal{M}$, alors $e \leq e'$ ou $e' \leq e$;*

(ii) c est clos par communication, c'est-à-dire que, pour tous $p, q \in \mathcal{P}$ et $m \in \mathcal{M}$, $|c|_{p!q(m)} = |c|_{q?p(m)}$;

(iii) c ne rajoute pas d'autre ordre que la causalité locale et celle des messages, c'est-à-dire que la relation $\leq$ est exactement $(\ll \bigcup_{p\in\mathcal{P}}(\leq \cap E_p^2))^*$.

Remarquons que la condition d'ordre total pour les événements liés à l'envoi ou à la réception d'un même message est utile uniquement pour des raisons techniques. En effet, cette condition permet de retrouver la causalité $\ll$ liée aux messages, en appariant le i-ième envoi d'un message de type m à la i-ième réception d'un message de même type. Nous aurions pu, de manière équivalente et comme nous l'avons fait dans [Gazagnaire 07b], rajouter aux pomsets une relation bijective $\ll$, attachée aux messages. Cependant, nous avons préféré rester, tout au long de ce document, dans le cadre des pomsets.

Nous munissons les MSC causaux de la composition des pomsets définie dans la partie 1.3.1 associée à une relation de dépendance (et donc réflexive et symétrique) comme pour les traces de la partie 1.2.1. Plus précisément, étant donné une relation de dépendance D, nous définissons la composition de deux MSC causaux de la même manière que la loi de composition des pomsets $\circledcirc_D$. L'ensemble de MSC causaux, muni de la loi $\circledcirc_D$ et du MSC vide ε, forme un monoïde noté $\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)$ qui est inclus dans $\mathbb{P}(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}},D)$.

Définition 2.2 (HMSC causal) *Soit $(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}},D)$, un alphabet concurrent localisé. Un* HMSC *(D-)causal est une expression rationnelle de $\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)$.*

Enfin, étant donné un HMSC causal, nous notons $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\alpha)$, l'ensemble des MSC causaux engendrés par α. L'ensemble des extensions visuelles d'un MSC causal c, noté $\text{MSC}(c)$, est l'ensemble des MSC qui sont des extensions d'ordre de ce MSC. La figure 2.2 montre un exemple d'extensions visuelles pour un MSC causal M. La boîte rectangulaire sur le processus p de M, représente des événements qui ne sont pas causalement reliés. Remarquons qu'une partie de cette extension apparaît dans le norme Z.120 des MSC sous le nom de "co-région", mais n'est jamais définie et utilisée dans leur sémantique classique.

Cette construction se généralise aisément aux langages. Ainsi, nous notons $\text{MSC}(\alpha)$, l'ensemble des extensions visuelles des MSC causaux engendrés par α.

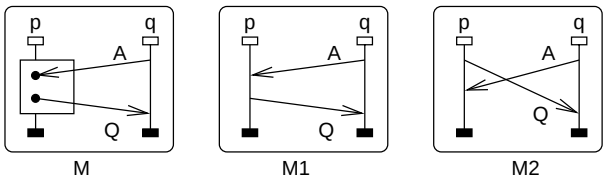


FIG. 2.2 – *A gauche, un* MSC *causal. Au centre et à droite, ses deux extensions visuelles.*

A partir de ces définitions, nous pouvons remarquer qu'un HMSC causal α permet de définir de manière syntaxique (au moins) trois langages différents, à savoir :

1. Le langage des linéarisations : $Lin(\alpha) \subseteq \Sigma^{\star}_{\mathcal{P},\mathcal{M},\mathcal{I}}$;
2. Le langage des MSC : $\text{MSC}(\alpha) \subseteq \mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})$;
3. Le langage des MSC causaux : $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\alpha) \subseteq \mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)$.

Nous verrons par la suite que différentes analyses seront possibles en fonction du langage qui nous intéresse.

Remarquons aussi que les HMSC causaux peuvent être facilement étendus aux CHMSC causaux, en reprenant les définitions de la partie 1.3.3.

Finalement, pour conclure cette partie, nous donnons, dans la figure 2.3, un exemple de spécification partielle du protocole du bit alterné, modélisé à l'aide d'un HMSC causal. Le modèle de haut niveau de ce protocole est donné par l'expression rationnelle $M1M2^{\star}M3$, associée à la relation d'indépendance :

$$I = \{(p!q(data0), p(inv)), (p?q(ack0), p(inv)), (q!p(ack0), q(inv)), (q?p(data0), q(inv))\}$$

Ce protocole permet d'assurer la fiabilité d'un échange de données entre deux processus p et q, en retransmettant régulièrement les messages qui n'ont pas été acquittés. Plus précisément, les données à envoyer sont d'abord coupées en plusieurs paquets qui vont s'insérer dans les messages *data* que p envoie à q (ces données n'ont pas été représentées sur la figure). Ensuite, p rajoute au premier message à envoyer un bit de contrôle. Puis, tant que q ne renvoie pas un accusé de réception étiqueté par ce même bit de contrôle, p transmet plusieurs fois ce nouveau message à q. Enfin, lorsque p reçoit l'accusé de réception en provenance de q, étiqueté par le bon bit, il recommence le même procédé avec le message suivant de données, mais en inversant le bit de contrôle qu'il va joindre à celui-ci. Pour des raisons de concision, nous n'avons modélisé que l'envoi de données avec le bit de contrôle 0.

Le MSC causal, représenté en bas à gauche de cette figure, correspond à l'exécution $M1M2M2M3$ du modèle. Le MSC représenté en bas à droite est une extension visuelle de ce MSC causal. Remarquons que le comportement engendré n'est pas finiment engendré, puisque les différents messages peuvent se croiser et les différentes phases du protocole s'entrelacer.

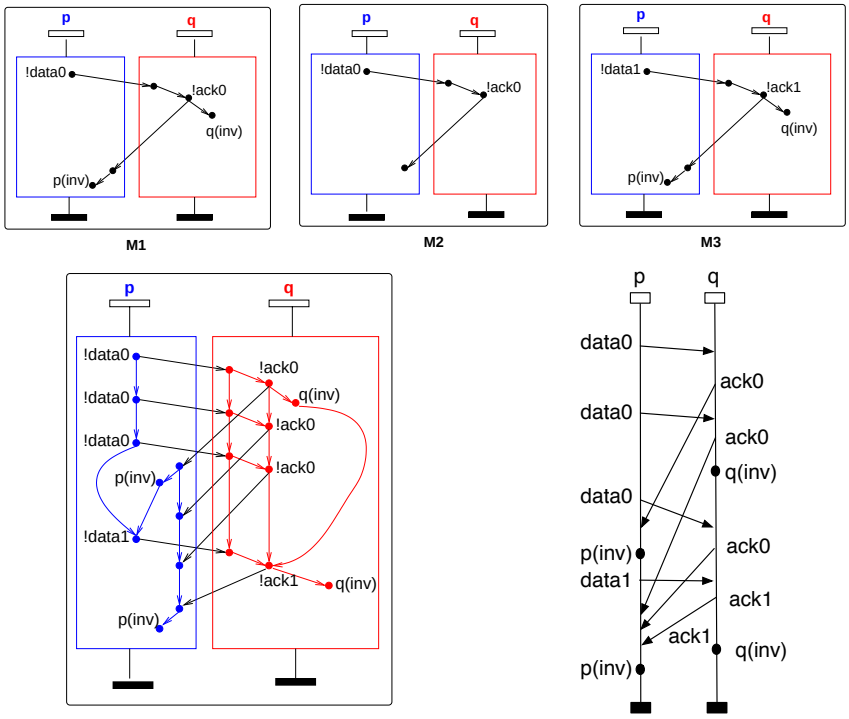


FIG. *2.3 – Le* HMSC *causal* $M1M2^{\star}M3$ *permet de décrire le protocole du bit alterné.*

2.2 Mise en œuvre vers des modèles communicants

Dans cette partie, nous présentons une technique de mise en œuvre des HMSC causaux dits réguliers par des réseaux bornés d'automates communicants. Cette partie s'organise de la manière suivante : tout d'abord, la partie 2.2.1 donne les principaux résultats concernant la mise en œuvre de HMSC et de CHMSC par des réseaux d'automates communicants. Ensuite,

dans la partie 2.2.2 nous présentons la restriction syntaxique des HMSC causaux réguliers et nous présentons le résultat de mise en œuvre de ces modèles par des réseaux d'automates communicants, résultat que nous démontrons, finalement, dans la partie 2.2.3.

2.2.1 Etat de l'art

Nous présentons, dans cette partie, les résultats principaux concernant la mise en œuvre de HMSC et de cHMSC par des réseaux d'automates communicants. Nous avons choisi de présenter ces résultats suivant deux axes. Premièrement, la mise en œuvre peut être vue comme la projection, sur chaque processus, des comportements globaux définis par le modèle initial. Deuxièmement, la mise en œuvre peut être vue comme une technique (très utilisée en algorithmique répartie) qui consiste à ajouter des données dans les messages envoyés afin de répartir le contrôle et de synchroniser les processus impliqués.

Avant de voir plus en détail les résultats, donnons quelques noms de classes de HMSC qui vont nous intéresser pour la mise en œuvre et dont les définitions seront données par la suite : (c)HMSC est l'ensemble des HMSC et des cHMSC, gc-HMSC est l'ensemble des HMSC globalement coopératifs, r-HMSC est l'ensemble des HMSC réguliers et lc-HMSC est l'ensemble des HMSC dits "à choix local", dont nous parlerons brièvement par la suite. Un réseau d'automates communicants est abrégé en RAC, et un RAC est dit sans blocage si tous les états globaux accessibles sont co-accessibles.

Nous allons donner les différents résultats, classés suivant les techniques algorithmiques mises en jeu et suivant la politique de routage dont dispose le réseau sous-jacent.

La première politique de routage, FIFO, fait l'hypothèse que le canal de communication séparant deux processus peut être considéré comme une file : le premier message entré est le premier message sorti. C'est le cas lorsque l'on se place sur des réseaux où les délais de communication sont très faibles et où le routage est relativement statique. C'est aussi le cas sur des réseaux plus complexes mais où l'on s'appuie sur un ensemble de protocoles de bas niveaux (comme TCP/IP sur internet) qui gère le routage et assure un ordre correct de délivrance aux messages.

La seconde politique de routage, weak-FIFO, fait l'hypothèse que ces canaux permettent à des messages ayant des types différents de se dépasser. C'est le cas en pratique lorsque l'on utilise Internet, un type donné de messages représentant, par exemple, une session ouverte entre deux utilisateurs. Pour deux sessions différentes, les messages correspondants n'ont aucune raison de ne pas se doubler : les chemins de routage sont différents et il n'existe pas, a priori, de raisons de forcer leur synchronisation lors de la réception. Par contre, au sein d'une même session, il est nécessaire que les paquets arrivent dans l'ordre dans lequel ils ont été émis.

Mise en œuvre par projection

Il existe plusieurs façons de mettre en œuvre un HMSC par un réseau d'automates communicants. La plus intuitive consiste à projeter le langage du HMSC sur chacune de ses instances, puis à faire communiquer les différents automates obtenus. La question à se poser est alors la suivante : le langage du réseau d'automates communicants obtenu par projection locale est-il le même que le langage de linéarisation du HMSC initial ? C'est à dire, plus formellement,

étant donné un HMSC α de REX($\mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$), est-ce que $Lin(\alpha) = \mathcal{L}(\|_{p\in\mathcal{P}}\ \pi_{\Sigma_p}(\alpha))$? (avec $\pi_{\Sigma_p}(\alpha)$, l'automate structurellement semblable à la projection de α sur Σ_p).

La figure 2.4 donne un exemple d'un HMSC et du réseau d'automates communicants obtenu par projection locale. Dans ce cas particulier, les langages générés par les deux structures sont identiques. Par la suite, nous allons voir que ce n'est pas toujours vrai, et que, de plus, dans le cas général, il n'est pas possible de décider de cette égalité.

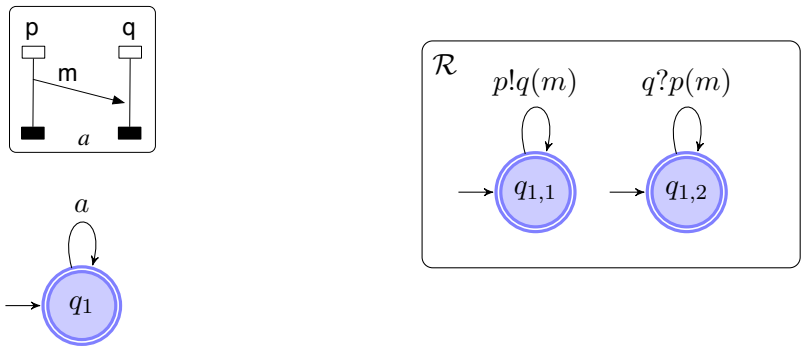


FIG. 2.4 – *A gauche, l'automate correspondant au* HMSC $a^\star$*, qui envoie un nombre non borné de messages* m *entre* p *et* q*. A droite, le réseau d'automates communicants résultant de la projection de ce* HMSC *sur chaque processus. Dans cet exemple, les deux structures génèrent le même langage.*

Nous donnons maintenant les principaux résultats concernant la mise en œuvre de HMSC et de CHMSC, en fonction des hypothèses choisies pour la politique de routage des messages entre les différents processus.

Hypothèse FIFO Les premiers résultats sur la mise en œuvre de HMSC dans le cas où les messages circulant entre les automates du réseau ne peuvent pas se croiser sur un même canal, ont été proposés par Alur, Etessami et Yannakakis dans [Alur 00, Alur 01]. Ils ont ainsi montré qu'il était indécidable de savoir si le langage du réseau d'automates communicants était le même que celui du HMSC initial.

Cependant, ils ont montré que, si le HMSC initial est régulier, alors savoir si le réseau d'automates communicants obtenu par projection locale est sans blocage et génère le même langage, est un problème qui est dans EXPSPACE. Un peu plus tard, Lohrey a montré, dans [Lohrey 02], que ce problème était EXPSPACE-complet et que ce résultat pouvait se généraliser aux HMSC globalement coopératifs.

Enfin, Ben-Abdallah et Leue ont introduit dans [BenAbdallah 97] la classe des HMSC à choix local, où le contrôle de l'exécution globale est à chaque moment localisé sur un seul processus. Hélouët et Jard ont montré dans [Hélouët 00a] que, savoir si un HMSC à choix local générait le même langage que le réseau d'automates communicants obtenu par projection locale et si celui-ci est sans blocage, est décidable. Il suffit en effet de tester son appartenance à une sous-classe syntaxique des HMSC dont les membres s'appellent les "HMSC reconstructibles", ce qui peut se faire en temps polynomial. Les preuves des résultats évoqués précédemment s'adaptent aisément aux CHMSC.

Le tableau de la figure 2.5 résume ces résultats. A priori, le problème de l'indécidabilité de

la mise en œuvre sans données des HMSC à choix local par un réseau quelconque d'automates communicants (qui peut donc avoir des blocages) reste ouvert.

↱ **sans** données	RAC	RAC sans blocage
(c)HMSC	Indécidable [Alur 00]	Indécidable [Lohrey 02]
gc-(c)HMSC	Indécidable [Alur 01]	EXPSPACE-c [Lohrey 02]
lc-(c)HMSC	?	P [Hélouët 00a]
r-(c)HMSC	Indécidable [Alur 01]	EXPSPACE-c [Alur 01, Lohrey 02]

FIG. *2.5 – Résultats de décidabilité pour le problème de la mise en œuvre de (c)HMSC par un réseau d'automates communicants FIFO, sans ajout de données.*

Hypothèse weak-FIFO Une autre série de résultats concerne les réseaux d'automates communicants avec des canaux de communication qui ne respectent pas forcément l'ordre FIFO. L'hypothèse weak-FIFO assouplit l'hypothèse FIFO en autorisant le croisement de deux messages avec des types différents. De manière plus pragmatique, ces réseaux possèdent une file FIFO associée à chaque nom possible de message.

Dans ce cadre, Morin a montré dans [Morin 02] que le problème de la mise en œuvre restait indécidable pour la classe des HMSC, mais que cela devenait décidable dès lors qu'on se restreignait aux HMSC globalement coopératifs. La preuve utilise une réduction vers le problème de l'atteignabilité dans les réseaux de Petri, dont la caractérisation de la complexité reste un problème ouvert (et est conjecturée être non-élémentaire, d'après [Mayr 81, Reutenauer 90]).

De la même manière, le problème reste décidable lorsque l'on veut, en plus, savoir si le réseau d'automates communicants obtenu par projection locale est sans blocage. Plus précisément, Baudru et Morin ont montré dans [Baudru 03] que le problème était EXPSPACE-complet.

Enfin, un autre problème du même genre est étudié par Caillaud et al. dans [Caillaud 00]. Ils y montrent que l'équivalence entre la fermeture par préfixe d'un langage de HMSC et le langage d'un réseau de Petri dit distribuable est indécidable. Les réseaux de Petri distribuables, sont quant à eux, équivalents aux réseaux d'automates communiquant par file weak-FIFO ([Caillaud 99]).

De même que pour le cas de l'hypothèse FIFO, les preuves des résultats évoqués précédemment s'adaptent aisément aux CHMSC. Le tableau de la figure 2.6 résume ces résultats.

↱ **sans** données	RAC wFIFO	RAC wFIFO sans blocage
(c)HMSC	Indécidable [Morin 02]	Indécidable [Morin 02]
gc-(c)HMSC	Décidable [Morin 02]	EXPSPACE-c [Baudru 03]
r-(c)HMSC	Décidable [Morin 02]	EXPSPACE-c [Baudru 03]

FIG. *2.6 – Résultats de décidabilité pour le problème de la mise en œuvre de (c)HMSC par un réseau d'automates communicants wFIFO, sans ajout de données.*

Mise en œuvre en ajoutant des données

La deuxième manière de mettre en œuvre un HMSC par un réseau d'automates communicants consiste à rajouter des données dans les messages, qui vont permettre de répartir le contrôle global, défini dans le HMSC, entre les différents automates communicants du réseau. Plus précisément, étant donné un HMSC α dans $\mathrm{REX}(\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I}))$, on cherche à trouver un ensemble fini $\mathcal{D}$ de données pour lequel il existe un HMSC $\widehat{\alpha}$ dans $\mathrm{REX}(\mathbb{MSC}(\mathcal{P},\mathcal{M}\times\mathcal{D},\mathcal{I}))$ et un réseau d'automates communicants $\langle S_i, (\widehat{\mathcal{A}}_{\mathcal{P}})_{p\in\mathcal{P}}\rangle$ à états finaux locaux, tels que les deux structures aient le même langage et que α soit l'image de $\widehat{\alpha}$ par un morphisme qui efface les données ajoutées. Plus formellement, nous cherchons à avoir $\mathcal{L}(\widehat{\alpha}) = \mathcal{L}(\langle S_i, \|_{p\in\mathcal{P}}\ \widehat{\mathcal{A}}_p\rangle)$ et $\eta(\widehat{\alpha}) = \alpha$, où η est le morphisme défini par $\eta(p!q(m,d)) = p!q(m)$, $\eta(p?q(m,d)) = p?q(m)$ et $\eta(p(a)) = p(a)$, pour tout $p,q \in \mathcal{P}$, $m \in \mathcal{M}$, $d \in \mathcal{D}$ et $a \in \mathcal{I}$. Dans le cas où une mise en œuvre sans blocage existe, la proposition 2.10, que nous montrons dans la suite, permet de reformuler le problème de la mise en œuvre avec données de la manière suivante : existe-t-il un HMSC $\widehat{\alpha}$ et un ensemble d'états globaux S_i tels que $\mathcal{L}(\widehat{\alpha}) = \mathcal{L}(\langle S_i, \|_{p\in\mathcal{P}}\ \pi_{\Sigma_p}(\widehat{\alpha})\rangle)$ et $\eta(\widehat{\alpha}) = \alpha$?

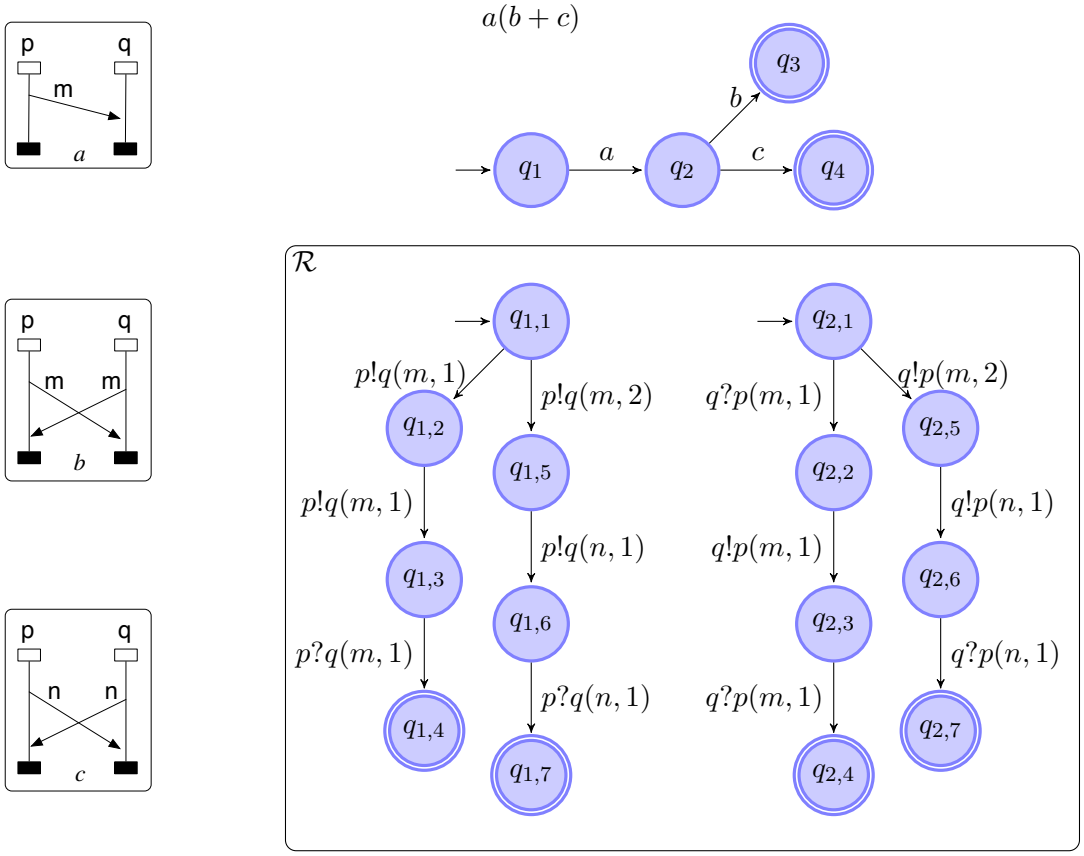


FIG. *2.7 – A gauche, et en haut à droite, un* HMSC $a(b+c)$ *qui ne peut pas être mis en œuvre sans données. En bas à droite, un réseau* $\mathcal{R}$ *d'automates communicants qui met en œuvre* $a(b+c)$ *en ajoutant une donnée* $d \in \{1,2\}$ *sur les messages, avec états initiaux et finaux locaux.*

La figure 2.7 illustre cette construction pour un HMSC. Plus précisément, cette figure

montre que le HMSC globalement coopératif $a(b+c)$ peut être mis en œuvre, sans blocage et avec ajout de données dans les messages. Ces données indiquent, ici, quelle branche du choix va être prise (si c'est un 1, c'est la branche correspondant à b qui est choisie, si c'est 2, c'est celle correspondant à c). Pour ce réseau, la projection locale donne un réseau d'automates communicants qui peut avoir des blocages (si p choisit d'envoyer m et q choisit d'envoyer n par exemple). Plus généralement, l'ajout de données facilite, dans de nombreux cas, la mise en œuvre de HMSC.

Hypothèse FIFO Nous donnons tout d'abord les résultats pour la mise en œuvre de HMSC par un réseau d'automates communiquant par files FIFO.

Tout d'abord, Genest, Muscholl, Seidl et Zeitoun ont montré dans [Genest 02b] que les HMSC à choix local peuvent toujours être mis en œuvre par un réseau d'automates communicants sans blocage. Pour ce faire, ils ont ajouté des données dans les messages qui contiennent des indications sur les choix futurs à effectuer par chaque processus.

Ensuite, Henriksen et al. ont montré dans [Henriksen 05] que tout ensemble régulier de MSC peut être mis en œuvre par un réseau d'automates communicants déterministes (mais avec blocage). Par conséquent, tous les HMSC et CHMSC réguliers peuvent être mis en œuvre.

Puis, Genest, Kuske et Muscholl ont montré dans [Genest 06a] que tout CHMSC globalement coopératif était équivalent (au sens de l'égalité des langages en ajoutant des données) à un réseau existentiellement borné d'automates communicants (qui peut comporter des blocages) : à chaque exécution d'un tel réseau, il existe une exécution générant le même ordre partiel reliant les événements et dont les canaux de communication restent bornés. Ainsi, tout HMSC globalement coopératif peut être mis en œuvre par un réseau d'automates communicants, avec ajout de données.

Ce dernier résultat, couplé au fait qu'il est indécidable de savoir si un HMSC est équivalent à un HMSC globalement coopératif ([Morin 02]), montre l'indécidabilité du problème de la mise en œuvre d'un HMSC par un réseau d'automates communicants. En effet, pour un HMSC, pouvoir être mis en œuvre équivaut à pouvoir être mis en œuvre par un réseau existentiellement borné d'automates communicants, et donc équivaut à être globalement coopératif.

Enfin, Baudru et Morin ont montré dans [Baudru 07] que tout ensemble régulier de MSC peut être mis en œuvre par un réseau borné d'automates non déterministes, communicants sans blocage et états finaux locaux, avec ajout de données et une hypothèse weak-FIFO. Cependant, la preuve reste (quasiment) identique dans le cas FIFO. Ainsi, tout HMSC ou CHMSC régulier peut être mis en œuvre sans blocage. A l'inverse, il est clair que tout langage de réseau borné d'automates communicants (sans blocage) est le langage d'un ensemble régulier de MSC, et donc d'un CHMSC régulier. Or, il est indécidable de savoir si un HMSC est régulier [Muscholl 99]. Au final, le problème de la mise en œuvre d'un CHMSC par un réseau borné et sans blocage d'automates communicants est donc indécidable.

Le tableau de la figure 2.8 résume les résultats précédents. A priori, le problème de la mise en œuvre avec données et sans blocage de HMSC globalement coopératifs en réseau d'automates communicants reste ouvert.

Hypothèse weak-FIFO Nous indiquons maintenant les résultats connus pour la mise en œuvre de CHMSC par des réseaux d'automates communicants par l'intermédiaire de canaux de communication weak-FIFO.

↱ **avec** données	RAC	RAC sans blocage
(c)HMSC	Indécidable [Genest 06a, Morin 02]	Indécidable [Baudru 07, Muscholl 99]
gc-(c)HMSC	Toujours [Genest 06a]	?
lc-(c)HMSC	Toujours [Genest 02b]	Toujours [Genest 02b]
r-(c)HMSC	Toujours^{Det} [Henriksen 05]	Toujours [Baudru 07]

FIG. *2.8 – Résultats de décidabilité pour le problème de la mise en œuvre de (c)*HMSC *par un réseau d'automates communicants FIFO, avec ajout de données.*

Tout d'abord, les résultats de Baudru et Morin ([Baudru 07]) énoncés précédemment sont bien évidemment toujours valides, puisque c'était l'hypothèse retenue dans leur travail. Ensuite, tous les résultats précédents utilisant une borne existentielle sont transposables dans le cas weak-FIFO en utilisant la technique suivante, définie par [Darondeau 07], qui augmente d'un facteur constant, la borne existentielle du modèle considéré. En effet, un cHMSC α dans REX($\mathbb{MSC}(\mathcal{P}, \mathcal{M}, \mathcal{I})$) peut être mis en œuvre dans le cas weak-FIFO s'il existe un cHMSC α' dans (c) REX($\mathbb{MSC}(\mathcal{P} \times \mathcal{M}, \{1\}, \mathcal{I})$) qui peut être mis en œuvre dans le cas FIFO, et tel que α' est exactement α auquel est ajouté, de manière non triviale, des processus dédiés à chaque type de message. Enfin, les résultats d'indécidabilité des problèmes de la mise en œuvre de (c)HMSC par des réseaux avec ou sans blocage, peuvent se montrer en utilisant les mêmes preuves que dans la partie précédente. Au final, le tableau de la figure 2.8 reste valable dans le cadre weak-FIFO.

2.2.2 HMSC causaux réguliers

Après avoir survolé rapidement les différentes techniques existantes de mise en œuvre de (c)HMSC en réseau d'automates communicants, nous nous focalisons maintenant sur la mise en œuvre de cHMSC causaux. Naturellement, ce problème est indécidable (proposition 2.3), car c'est déjà le cas pour les HMSC. Cependant, nous allons nous restreindre, dans la suite de ce chapitre, aux systèmes communiquant par files FIFO et dont les capacités sont bornées. Cela va nous permettre d'identifier une sous-classe syntaxique des cHMSC causaux, appelée cHMSC causaux *réguliers*, dont les membres ont un langage régulier de linéarisations et peuvent, par conséquent, être mis en œuvre, avec données, par des réseaux universellement bornés d'automates communicants (théorème 2.5). A l'inverse, nous identifions, dans la partie 2.3, une sous-classe de réseaux universellement bornés d'automates communicants, appelés automates mixtes car localement décomposables en automates asynchrones, que nous montrons être équivalents (à renommage près) aux cHMSC dits réguliers et *cohérents* (théorème 2.14).

Pour les systèmes parallèles bornés, interagissant par échange de messages, nous donnons donc, dans ce chapitre, un ensemble d'outils adéquats de description et d'analyse. Concernant la description, nous savons caractériser des classes de systèmes bornés qui mélangent communications par mémoire partagée et par échange de messages. Concernant l'analyse, nous pouvons utiliser les outils classiques définis sur les automates, même si, en général, la traduction d'un modèle de haut niveau vers un modèle opérationnel exprimé sous la forme d'automate a un coût exponentiel.

Indécidabilité du cas général

Tout d'abord, lorsque l'on considère la classe des cHMSC causaux, un grand nombre de questions sont indécidables. Ce n'est pas une surprise, car ces résultats d'indécidabilité sont directement hérités de ceux existant pour les cHMSC. Plus précisément :

Proposition 2.3 *Soient* $(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D)$*, un alphabet concurrent localisé et* α*, un (c)*HMSC *D-causal. Alors, il est indécidable de savoir si* α *peut être mis en œuvre par un réseau d'automates communicants :*

- *sans ajout de données ;*
- *sans ajout de données et sans blocage ;*
- *avec ajout de données ;*
- *avec ajout de données et sans blocage.*

Preuve: Les (c)HMSC sont des cHMSC causaux. Il suffit ensuite d'appliquer les résultats d'indécidabilité énoncés dans la partie 2.2.1. □

Ces premiers résultats négatifs, motivent la recherche de sous-classes décidables pour ce nouveau modèle de spécification. Ainsi, dans le cadre de ce chapitre, nous allons voir qu'il est possible, comme pour les (c)HMSC, d'identifier des sous-classes syntaxiques sur lesquelles le problème de la mise en œuvre, avec ajout de données dans les messages, devient décidable.

Régularité du langage de linéarisations

Afin de résoudre les problèmes d'indécidabilité évoqués précédemment, nous devons réduire l'expressivité des modèles considérés. Pour cela, nous proposons d'étendre la classe des cHMSC réguliers, en prenant en compte, pour chaque cycle et chaque processus, une propriété similaire à celles définies sur les expressions corationnelles de traces. Remarquons que pour obtenir une cohérence entre la politique de routage dans les réseaux d'automates communicants (définis comme étant FIFO dans le chapitre 1) et la politique de routage dans les cHMSC causaux, la définition de régularité impose que tous les MSC engendrés doivent être FIFO (et il suffit de l'imposer dans les expressions qui correspondent à des boucles et dans toutes les sous-expressions maximales sans boucles). La preuve est identique si l'on se place dans le cadre où les réseaux d'automates communicants ont une politique de routage weak-FIFO.

Définition 2.4 (régularité) *Soient* $(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D)$ *un alphabet concurrent localisé et* α *un c*HMSC *D-causal.* α *est dit* régulier *si :*

- *pour toute sous-expression* $\beta^\star$ *de* α*, tout* MSC $m \in \mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\beta)$ *vérifie les quatre propriétés suivantes :*
  - *(i) pour tout processus* p*,* $\Sigma_p(m)$ *est D-connexe ;*
  - *(ii) le graphe de communication de* m *est fortement connexe ;*
  - *(iii)* m *est FIFO ;*
- *(iv) pour toute sous-expression* δ *de* α *sans* $\star$*, tout* $m \in \mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\delta)$ *est FIFO.*

La figure 2.9 donne un HMSC causal régulier dont le langage de MSC n'est pas finiment engendré, et donc n'est pas le langage d'un HMSC régulier. La relation de dépendance choisie est telle que les deux messages commutent entre eux, mais pas avec les actions internes, ce

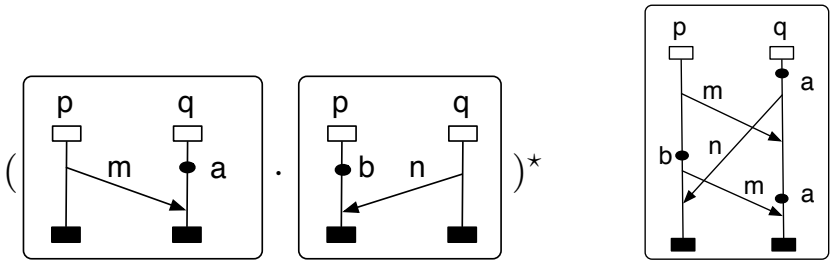


FIG. 2.9 – *Un* HMSC *causal* α*, dont les messages* m *et* n *peuvent commuter lors de la composition :* $I = \{(p!q(m), p?q(n)), (p?q(n), p!q(m)), (q?p(m), q!p(n)), (q!p(n), q?p(m))\}$ *et à droite un* MSC *dans* $\text{MSC}(\alpha)$.

qui fait que la projection de l'alphabet sur chaque processus est D-connexe. A droite de cette figure, est représenté un MSC engendré par ce modèle.

Remarquons que la définition précédente donne un algorithme CoNP immédiat pour décider si un CHMSC causal est régulier.

Finalement, cette sous-classe syntaxique possède deux intérêts principaux. En effet, comme pour les CHMSC réguliers et les expressions corationnelles de traces, les problèmes de la régularité du langage des linéarisations et de la mise en œuvre avec données deviennent décidables.

Théorème 2.5 *Soient* $(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D)$ *un alphabet concurrent localisé et* α *un c*HMSC *D-causal. Si* α *est régulier, alors :*

- *(Régularité)* $Lin(\alpha)$ *est régulier ;*
- *(Mise en œuvre)* α *peut être mis en œuvre, avec ajout de données, par une union finie de réseaux universellement bornés d'automates communicants déterministes et sans blocage.*

Nous démontrons ce théorème dans la partie suivante.

2.2.3 Preuve du Théorème 2.5 (régularité et mise en œuvre)

Régularité

Dans cette partie, nous démontrons le théorème 2.5, qui dit que tout CHMSC causal régulier peut être mis en œuvre avec données par une union finie de réseaux bornés et sans blocage d'automates communicants déterministes.

Un théorème semblable a été montré pour les HMSC réguliers par [Kuske 03] en utilisant la méthode suivante. Dans une première étape, le HMSC régulier est codé vers les expressions corationnelles de traces, à l'aide d'un encodage adéquat qui s'appuie sur la borne universelle de toute exécution. Dans une seconde étape, le l'expression corationnelle de trace est transformée en automate en utilisant les résultats de [Ochmanski 85]. Cette preuve a ensuite été adaptée dans [Genest 04b] pour le cas des CHMSC globalement coopératifs, en utilisant une traduction vers les traces corationnelles s'appuyant sur une borne existentielle. Cependant, dans le cas des HMSC causaux, trouver un tel encodage semble impossible à cause de l'entrelacement

irrégulier des événements (car les ordres locaux ne sont pas nécessairement cohérents avec D).

Pour cette raison, nous avons fait le choix de reprendre la preuve originale d'Ochmanski sur la régularité des langages corationnelles de traces ([Ochmanski 85]), en nous inspirant des preuves constructives données dans [Muscholl 99] ou [Genest 05].

Pour cette preuve, nous fixons un alphabet concurrent localisé $(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D)$ et, pour des raisons de simplicité, un HMSC D-causal régulier α et nous montrons que nous pouvons construire un $\Sigma^{\star}_{\mathcal{P},\mathcal{M},\mathcal{I}}$-automate $\mathcal{A}_{lin}$ qui reconnait $Lin(\alpha)$. La même construction permet de donner ce même résultat lorsque l'on permet de couper les messages, c'est-à-dire pour des CHMSC D-causaux réguliers. Nous indiquons, lorsque cela est nécessaire, les endroits où la traduction entre les preuves n'est pas immédiate.

Pour cette preuve, nous avons besoin de représenter l'expression rationnelle α par un automate. Rappelons que $\Gamma(\alpha)$ est l'ensemble des MSC causaux finis utilisés pour construire α. Nous pouvons alors construire un $\Gamma(\alpha)^{\star}$-automate $\mathcal{A}_\alpha = (S, \rightarrow, \Gamma(\alpha), S_i, S_f)$ ne contenant pas d'epsilon transition, et tel que $\mathcal{L}(\alpha) = \mathcal{L}(\mathcal{A}_\alpha)$ et $|S| = O(|\alpha|)$ ([Thompson 68]). Il est immédiat de voir que si α est régulier, alors tout cycle de $\mathcal{A}_\alpha$ est étiqueté par un mot D-connexe. On dira alors que $\mathcal{A}_\alpha$ est D-connexe, et on parlera indistinctement de MSC D-connexes appartenant au langage d'une sous-expression $\beta^{\star}$ de α ou de cycles D-connexes de $\mathcal{A}_\alpha$: α et $\mathcal{A}_\alpha$ seront dits *structurellement semblables*.

Nous établissons maintenant une série de lemmes techniques.

Lemme 2.6 *Soit un* MSC $m \in \mathcal{L}(\alpha)$ *étiquetant un chemin* $\rho = \theta_1 \dots \theta_{|\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}|}$ *de* $\mathcal{A}_\alpha$ *où chaque* $\theta_i = s_{i,0} \xrightarrow{m_{I,1}} \dots \xrightarrow{m_{i,\ell_i}} s_{i,0}$ *est un cycle (ces cycles n'étant pas forcément contiguës). Supposons, de plus, que tous les ensembles de* MSC *causaux* $\Gamma(\theta_i) = \{m_{i,1}, \dots, m_{i,\ell_i}\}$, *pour* $i = 1 \dots |\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}|$, *sont égaux. Enfin, soient* e *et* e' *deux événements appartenant aux* MSC *qui étiquettent respectivement* θ_1 *et* $\theta_{|\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}|}$. *Alors,* $e \leq_m e'$.

Preuve: Notons Σ_p l'ensemble des actions de $\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$ qui sont localisés sur le processus p et $\Sigma_p(\theta_i)$ l'ensemble des actions du MSC qui étiquette θ_i qui sont localisés sur le processus p. Nous pouvons alors considérer deux cas :

— **Cas (i)** : $loc(\lambda(e)) = loc(\lambda(e'))$.

Soit $p = loc(\lambda(e))$. Rappelons que si α est régulier, alors pour chaque θ_i, $\Sigma_p(\theta_i)$ est D-connexe. Remarquons aussi que $\Sigma_p(\theta_1) = \dots = \Sigma_p(\theta_{|\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}|}) \subseteq \Sigma_p$. Par conséquent, nous pouvons trouver un ensemble d'événements $\{e_j\}_{j=1,\dots,t}$, où $t \leq |\Sigma_p| - 2$, chaque e_j étant dans θ_{j+1} et tels que $\lambda(e)\ D_p\ \lambda(e_1) D_p \dots\ D_p\ \lambda(e_t)\ D_p\ \lambda(e')$. Donc $e \leq e_1 \leq \dots \leq e_t \leq e'$.

— **Cas (ii)** : $loc(\lambda(e)) \neq loc(\lambda(e'))$.

Soient $p_1 p_2 \dots p_t$ un chemin de $loc(\lambda(e))$ à $loc(\lambda(e'))$ dans le graphe de communication de m, où $p_1 = loc(\lambda(e))$, $p_t = loc(\lambda(e'))$. De la même manière que pour le cas (i), il est facile de voir que l'on peut choisir des événements e_i, f_i dans θ_{u_i}, pour $i = 1, \dots, t-1$, tels que pour chaque i, $u_i = |\Sigma_p(\theta_1)| + \dots + |\Sigma_p(\theta_i)| \leq |\Sigma_p|$, $loc(e_i) = p_i$, $loc(f_i) = p_{i+1}$ and $e_i \ll f_i$. Au final, $e \leq e_1 \ll f_1 \leq e_2 \ll f_2 \leq \dots \leq e_{t-1} \ll f_{t-1} \leq e'$. □

Soit $\rho = s_0 \xrightarrow{m_1} \dots \xrightarrow{m_\ell} s_\ell$ un chemin de $\mathcal{A}_\alpha$. Notons $m_i = (E_i, \lambda_i, \leq_i)$ pour chaque $i = 1, \dots, \ell$ et $(E, \lambda, \leq) = m_1 \circledcirc_D \dots \circledcirc_D m_\ell$. Une *configuration* de ρ est un sous-ensemble de E clos par précédence. Soit C, une configuration de ρ. Un C-sous-chemin de ρ est sous-chemin $\varrho = s_u \xrightarrow{m_{u+1}} \dots \xrightarrow{m_{u'}} s_{u'}$ de ρ étiqueté par $m_{u+1} \dots m_{u'}$, tel que $C \cap E_i \neq \emptyset$ pour chaque $i = u+1, \dots, u'$ et maximal pour cette propriété. Etant donné un tel C-sous-chemin ϱ, nous définissons son *C-résidu* comme étant l'ensemble $(E_{u+1} \cup E_{u+2} \cup \dots \cup E_{u'}) - C$. La figure 2.10 illustre ces notions. Chaque MSC causal est représenté par un rectangle. Les événements d'une configuration C sont représentés par des disques noirs, les événements qui ne sont pas dans C par des disques blancs, et la configuration est représentée par un polygone coloré aux bords en pointillés. Les deux C-sous-chemins représentés sur la figure 2.10 sont des séquences de MSC partiels que l'on peut construire à partir des événements apparaissant dans C. Le lemme suivant donne une borne sur le nombre de C-sous-chemins que peut contenir une configuration C.

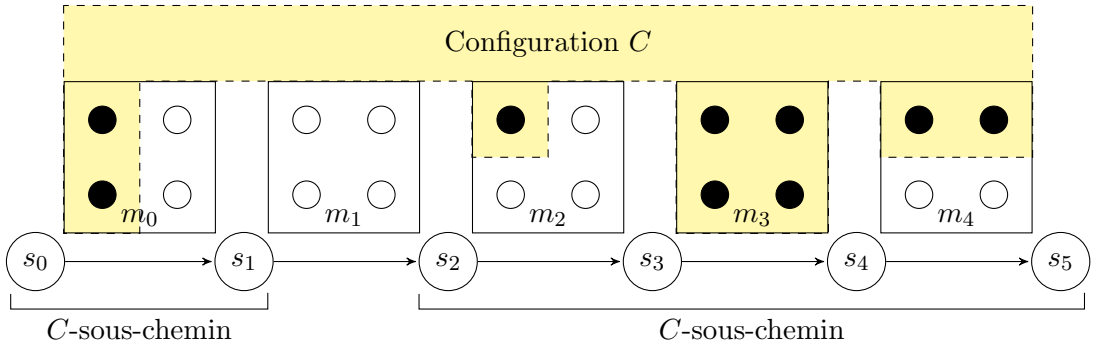


FIG. *2.10 – Les événements des C-sous-chemins sont représentés par des disques noirs. Les événements des C-résidus sont représentés par des disques blancs.*

Lemme 2.7 *Soient ρ, un chemin de $\mathcal{A}_\alpha$ et C, une configuration de ρ. Alors :*

(i) Le nombre de C-sous-chemins de ρ est au plus de $K_{expr} = |\alpha| \cdot |\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}| \cdot 2^{|\Gamma(\alpha)|}$, où $|\alpha|$ est le nombre de lettres dans l'expression rationnelle α et $|\Gamma(\alpha)|$ est le nombre de MSC *causaux distincts apparaissant dans α ;*

(ii) Soit ϱ, un C-sous-chemin de $\mathcal{A}_\alpha$. Alors, le nombre d'événements dans le C-résidu de ϱ est au plus de $K_{res} = |\alpha| \cdot |\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}| \cdot 2^{|\Gamma(\alpha)|} \cdot \max\{|E_c| \mid c \in \Gamma(\alpha)\}$ où $|E_c|$ est le nombre d'événements apparaissant dans le MSC *causal c.*

Preuve:

(i) Montrons ce résultat par contradiction. Supposons $K > |\alpha| \cdot |\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}| \cdot 2^{|\Gamma(\alpha)|}$. Posons $\ell = \Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}| \cdot 2^{|\Gamma(\alpha)|}$. Tout d'abord, nous pouvons trouver $\ell + 1$ C-sous-chemins ayant les mêmes états finaux. Soient $i_1 < i_2 < \dots < i_{\ell+1}$, les indices de ces états. Pour $h = 1, \dots, \ell$, soit θ_h le sous-chemin de ρ ayant pour origine n_{i_h} et pour état final $n_{i_{h+1}}$. Ensuite, parmi ces $\ell + 1$ C-sous-chemins, nous pouvons en trouver $|\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}|$, notés θ_{j_1}, …, $\theta_{j_{|\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}|}}$ pour $j_1 < \dots < j_{|\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}|}$, tels que $\Gamma(\theta_{j_1}) = \dots = \Gamma(\theta_{j_{|\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}|}})$. Finalement, prenons un événement e dans le MSC étiquetant θ_{j_1} tel que $e \notin C$. Un tel e existe, car aucun des événements du premier MSC causal apparaissant dans θ_{j_1} n'est dans C. Prenons ensuite un événement e' dans $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\theta_{j_{|\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}|}})$ tel que $e' \in C$. Nous

pouvons ensuite appliquer le lemme 2.6 pour obtenir que $e < e'$. Or, par définition, C est clos par précédence, ce qui est une contradiction.

(ii) Soient $\varrho = s_i \xrightarrow{m_{i+1}} \dots \xrightarrow{m_{i'}} s_{i'}$ et $E_i = \widehat{E}_{m_i} - C$, pour $j = 1, \dots, n$. En utilisant des arguments similaires au point (i), il est facile de montrer que, parmi $\widehat{E}_1$, …, $\widehat{E}_n$, au plus $|\alpha| \cdot |\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}| \cdot 2^{|\Gamma(\alpha)|}$ sont vides. Ce qui conclut la preuve.

□

Nous pouvons maintenant définir le $\Sigma^\star_{\mathcal{P},\mathcal{M},\mathcal{I}}$-automate $\mathcal{A}_{lin} = (S', \Rightarrow, \Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, S'_i, S'_f)$ tel que $\mathcal{L}(\mathcal{A}_{lin}) = Lin(\alpha)$. Pour cela, nous fixons tout d'abord les constantes K_{expr} et K_{res} telles que définies dans le lemme 2.7, et nous utilisons la convention suivante : si $m = [E, \lambda, \leq]$ est un MSC causal et E' un sous-ensemble de E, alors la restriction de m à E' est abusivement notée $m' = [E', \lambda, \leq]$. De plus, étant donné un chemin ρ de $\mathcal{A}_\alpha$, qui est, rappelons-le, le $\Gamma(\alpha)^\star$-automate ayant le même langage que α dans le monoïde $(\Gamma(\alpha)^\star, \cdot)$, nous notons $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\rho)$ l'interprétation du mot $u \in \Gamma(\alpha)^\star$ étiquetant ρ dans $\mathbb{MSC}((\mathcal{P}, \mathcal{M}, \mathcal{I}), D)$.

Nous donnons, tout d'abord, une idée intuitive de la construction de $\mathcal{A}_{lin}$. Pour un mot σ dans $\Sigma^\star_{\mathcal{P},\mathcal{M},\mathcal{I}}$, $\mathcal{A}_{lin}$ va deviner un chemin acceptant ρ de $\mathcal{A}_\alpha$ et vérifier que σ appartient à l'ensemble des linéarisation des MSC causaux engendrés par ρ, c'est-à-dire $Lin(\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\rho))$. Pour cela, après avoir lu un préfixe σ' de σ, $\mathcal{A}_{lin}$ mémorise une séquence de C-sous-chemins de ρ à partir desquels σ' a été "linéarisé". En utilisant le lemme 2.7, il est clair qu'à chaque moment, il faut retenir au plus K_{expr} tels C-sous-chemins. De plus, pour chaque C-sous-chemin, il faut conserver un nombre borné d'informations, qui est conservé dans une structure appelée "segment".

Segment Un *k-segment* est un k-uplet $(n_1, \widehat{\Sigma}_1, W_1, n'_1) \dots (n_k, \widehat{\Sigma}_k, W_k, n'_k)$, où pour tout $i \in \{1, \dots, k\}$, $n_i, n'_i \in S$, $\widehat{\Sigma}_i$ est un sous-ensemble non vide de $\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$, et W_i est un MSC causal dont le nombre d'événements est borné par K_{res}. De plus, pour tout $i \in \{1, \dots, k-1\}$, pour tout $i \leq j \leq k$, pour tout $\sigma \in \widehat{\Sigma}_j$, σ commute avec toutes les lettres étiquetant des événements de W_i, c'est-à-dire pour tout $e \in E_{W_i}$, $\sigma \; I \; \lambda_{W_i}(e)$.

De manière intuitive, un 1-segment $(n, \widehat{\Sigma}, W, n')$ se souvient d'un C-sous-chemin ρ de $\mathcal{A}_\alpha$ qui commence en n et se termine en n'. De plus, $\widehat{\Sigma}$ est l'ensemble des actions qui étiquettent les événements de $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\rho)$ qui ont déjà été linéarisés. Enfin, W est la restriction de $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\varrho)$ à l'ensemble des événements qui n'ont pas encore été linéarisés.

Construction de $\mathcal{A}_{lin}$ Nous définissons enfin $\mathcal{A}_{lin} = (S', \Rightarrow, \Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, S'_i, S'_f)$ de la manière suivante :

- Comme énoncé plus haut, S' est l'ensemble des K_{expr}-segments ;
- L'état initial est $S'_i = \{\varepsilon\}$, où ε est la séquence vide ;
- Un état est final si, et seulement si, il est composé d'un unique segment $\theta = (n, \widehat{\Sigma}, \epsilon, n')$ tel que $n \in S_i$ et $n' \in S_f$ (et $\widehat{\Sigma}$ est un quelconque sous-ensemble non vide de $\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$) ;
- La relation de transition $s \Rightarrow \hat{s}$ de $\mathcal{A}_{lin}$ est la plus petite relation satisfaisant les 5 conditions suivantes (où $s = \theta_1 \dots \theta_k \theta_{k+1} \dots \theta_\ell$, avec $\theta_i = (n_i, \widehat{\Sigma}_i, W_i, n'_i)$) :
  1) "Créer un nouveau segment" Si $n \xrightarrow{m} n'$ dans $\mathcal{A}_\alpha$, notons e un élément minimal de m (qui existe car $\mathcal{A}_\alpha$ est sans espsilon transitions). Notons alors $\theta = (n, \{\lambda_m(e)\}, m - \{e\}, n')$. Si $\hat{s} = \theta_1 \dots \theta_k \theta \theta_{k+1} \dots \theta_\ell$ est un K_{expr}-segment, alors $s \Rightarrow \hat{s}$;

2) "Ajouter au début d'un segment" Si $n \xrightarrow{m} n'$ dans $\mathcal{A}_\alpha$, notons e un élément minimal de m (qui existe car $\mathcal{A}_\alpha$ est sans espsilon transitions) et si $n' = n_{k+1}$, notons $\hat{\theta} = (n, \{\lambda_m(e)\} \cup \Gamma_{k+1}, (m - \{e\}) \circledcirc_D W, n'_{k+1})$. Si $\hat{s} = \theta_1 \dots \theta_k \hat{\theta} \theta_{k+2} \dots \theta_\ell$ est un K_{expr}-segment, alors $s \Rightarrow \hat{s}$;
3) "Ajouter à la fin d'un segment" Si $n \xrightarrow{m} n'$ dans $\mathcal{A}_\alpha$, notons e un élément minimal de m (qui existe car $\mathcal{A}_\alpha$ est sans espsilon transitions) et si $n = n'_k$, notons $\hat{\theta} = (n_k, \Gamma_k \cup \{\lambda_m(e)\}, W \circledcirc_D (m - \{e\}), n')$. Si $\hat{s} = \theta_1 \dots \theta_{k-1} \hat{\theta} \theta_{k+1} \dots \theta_\ell$ est un K_{expr}-segment, alors $s \Rightarrow \hat{s}$;
4) "Fusionner deux segments" Si $n \xrightarrow{m} n'$ dans $\mathcal{A}_\alpha$, notons e un élément minimal de m (qui existe car $\mathcal{A}_\alpha$ est sans espsilon transitions) et si $n = n'_k$ et $n' = n_{k+1}$, notons $\hat{\theta} = (n_k, \Gamma_k \cup \{\lambda_m(e)\} \cup \Gamma_{k+1}, W_k \circledcirc_D (m - \{e\}) \circledcirc_D W_{k+1}, n'_{k+1})$. Si $\hat{s} = \theta_1 \dots \theta_{k-1} \hat{\theta} \theta_{k+2} \dots \theta_\ell$ est un K_{expr}-segment, alors $s \Rightarrow \hat{s}$;
5) "Consommer un événement" Si W_k est non vide, notons e un élément minimal de W_k, $\hat{\theta} = (n_k, \Gamma_k \cup \{\lambda_{W_k}(e)\}, W_k - \{e\}, n'_k)$ et $\hat{s} = \theta_1 \dots \theta_{k-1} \hat{\theta} \theta_{k+1} \dots \theta_\ell$. Alors, $s \Rightarrow \hat{s}$.

Si l'on s'intéresse à des CHMSC causaux, la construction précédente est pratiquement inchangée. En effet, il suffit, alors, de rajouter l'état des canaux de communication à chaque composant d'un k-segment. Cet état, codé par une fonction qui associe à chaque élément (p, m, q) de $\mathcal{P} \times \mathcal{M} \times \mathcal{P}$ un entier, indique le nombre de messages m envoyés entre p et q qui ont déjà été linéarisés pour le C-chemin courant. Il faut alors s'appuyer sur la propriété de sûreté, qui dit que, pour tout cycle, les nombres d'envois et de réceptions de messages sont identiques, pour montrer que l'état des canaux est borné. Le nombre d'états de $\mathcal{A}_{lin}$ est modifié par un facteur $(|b|^{|\mathcal{P}|^2 \cdot |\mathcal{M}|})^{K_{expr}}$, où b est la borne universelle du CHMSC D-causal sûr considéré et $|\mathcal{P}|^2 \cdot |\mathcal{M}|$ le nombre de canaux de communication.

Nous avons terminé de construire $\mathcal{A}_{lin}$. Il reste maintenant à prouver que cette construction est correcte, afin de compléter la preuve du théorème 2.5. C'est ce que nous faisons avec le lemme suivant.

Lemme 2.8 *$\sigma \in \Sigma^\star_{\mathcal{P},\mathcal{M},\mathcal{I}}$ est accepté par $\mathcal{A}_{lin}$ si, et seulement si, σ est dans $Lin(\alpha)$.*

Preuve: Soit $\sigma = a_1 a_2 \dots a_k$, un mot de $Lin(\alpha)$. Prenons ensuite un chemin acceptant ρ dans $\mathcal{A}_\rho$, tel que $\rho = n_0 \xrightarrow{m_1} \dots \xrightarrow{m_\ell} n_\ell$, tel que σ soit une linéarisation de $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I},)D)}(\rho)$. Nous avons donc $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\rho) = (E, \lambda, \leq)$ avec $E = \{e_1, e_2, \dots, e_k\}$ et $\lambda(e_i) = a_i$ pour $i = 1, \dots, k$. De plus, $e_i \leq e_j$ implique que $i \leq j$ pour tout i, j dans $\{1, \dots, k\}$. Considérons maintenant les configurations $C_i = \{e_1, \dots, e_i\}$, pour $i = 1, \dots, k$. Nous pouvons associer à chaque C_i un état s_i de $\mathcal{A}_{lin}$ de la manière suivante : considérons une configuration C_i fixée. Posons $\rho = \dots \varrho_1 \dots \varrho_2 \dots \varrho_h \dots$ où ϱ_1, ϱ_2, …, ϱ_h sont les C-chemins de ρ. Posons ensuite $s_i = \theta_1 \dots \theta_h$, où $\theta_j = (n_j, \widehat{\Sigma}_j, W_j, n'_j)$ avec n_j étant l'état initial de ϱ_j et $\widehat{\Sigma}_j$ l'ensemble des $\lambda(e)$ pour tous les événements e qui sont à la fois dans $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\varrho_j)$ et C_i. W_j le MSC causal issu de la restriction de $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\rho)$ au le C_i-résidu de ϱ_j. Finalement, n'_j est l'état final de ϱ_j.

Il est maintenant possible de vérifier que $\varepsilon \stackrel{a_1}{\Rightarrow} s_1 \dots s_{k-1} \stackrel{a_k}{\Rightarrow} s_k$ est un chemin acceptant de $\mathcal{A}_{lin}$. Inversement, étant donné un chemin acceptant de $\mathcal{A}_{lin}$, il est trivial de construire le chemin acceptant correspondant dans $\mathcal{A}$. □

Nous avons donc montré que le test syntaxique de régularité (à savoir que le graphe de communication est fortement connexe et que tout cycle est D-connexe) permettait de construire effectivement un automate $\mathcal{A}_{lin}$ qui reconnaît les linéarisations de $\mathcal{A}$. Nous donnons maintenant un résultat supplémentaire concernant la taille de l'automate $\mathcal{A}_{lin}$ construit.

Proposition 2.9 *Soit $(\Sigma_{\mathcal{M},\mathcal{P},\mathcal{I}}, D)$, un alphabet concurrent localisé. Pour tout* HMSC *D-causal régulier α, notons $m = \max\{|E_\gamma| \mid \gamma \in \Gamma(\alpha)\}$. Alors, le nombre d'états de l'automate qui reconnaît $Lin(\alpha)$ est borné par :*

$$\left(|\alpha|^2 \cdot 2^{|\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}|} \cdot 2^{|\alpha|\cdot|\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}|\cdot 2^{|\Gamma(\alpha)|}\cdot m}\right)^{|\alpha|\cdot|\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}|\cdot 2^{|\Gamma(\alpha)|}}$$

Preuve: Pour la complexité, il faut compter le nombre de K_{expr}-segments. En s'appuyant sur le lemme 2.7, on obtient le résultat escompté. □

Mise en œuvre

Rappelons le résultat montré dans la partie précédente : tout (c)HMSC causal régulier génère un ensemble régulier de MSC. Il suffit alors d'appliquer le résultat de [Baudru 07], disant que tout ensemble régulier des MSC peut être mis en œuvre avec données par un réseau borné et sans blocage d'automates communicants à états finaux locaux, pour obtenir que tout (c)HMSC causal régulier peut être mis en œuvre avec données par un réseau borné et sans blocage d'automates communicants à états finaux locaux. Nous introduisons maintenant une proposition permettant de décomposer ces réseaux d'automates communicants en une union finie de réseaux sans blocage d'automates communicants *déterministes*.

Proposition 2.10 *Soit $\mathbb{C}_0$, l'ensemble des réseaux sans blocage d'automates communicants, avec états initiaux et finaux locaux. Alors, pour tous les réseaux $(\mathcal{A}_p)_{p\in\mathcal{P}}$ et $(\mathcal{A}'_p)_{p\in\mathcal{P}}$ dans $\mathbb{C}_0$ tels que, pour tout $p \in \mathcal{P}$, $\mathcal{L}(\mathcal{A}_p) = \mathcal{L}(\mathcal{A}'_p)$ on a : $\mathcal{L}(||_{p\in\mathcal{P}}\,\mathcal{A}_p) = \mathcal{L}(||_{p\in\mathcal{P}}\,\mathcal{A}'_p)$.*

Preuve: Supposons que les langages $\mathcal{L}(||_{p\in\mathcal{P}}\,\mathcal{A}_p)$ et $\mathcal{L}(||_{p\in\mathcal{P}}\,\mathcal{A}'_p)$ soient différents. Supposons alors que l'on puisse trouver un mot u tel que $u \in \mathcal{L}(||_{p\in\mathcal{P}}\,\mathcal{A}_p)$ et $u \notin \mathcal{L}(||_{p\in\mathcal{P}}\,\mathcal{A}'_p)$. Tout d'abord, les réseaux de $\mathbb{C}_0$ étant à états initiaux et finaux locaux, nous pouvons, pour tout chemin acceptant ρ dans le réseau $(\mathcal{A}_p)_{p\in\mathcal{P}}$ et tout $p \in \mathcal{P}$, trouver un chemin acceptant ρ_p dans l'automate $\mathcal{A}_p$ tel que la projection du mot étiquetant ρ sur Σ_p soit exactement le mot étiquetant ρ_p. Ainsi, si u est dans $\mathcal{L}(||_{p\in\mathcal{P}}\,\mathcal{A}_p)$ alors, pour tout $i \in \{1,\ldots,|\mathcal{P}|\}$, $\pi_{\Sigma_i}(u) \in \mathcal{L}(\mathcal{A}_i)$. Ainsi, si u n'est pas dans $\mathcal{L}(||_{p\in\mathcal{P}}\,\mathcal{A}'_p)$ alors, il existe un $i \in \{1,\ldots,|\mathcal{P}|\}$ tel qu'aucun chemin acceptant de $\mathcal{A}'_i$ ne soit étiqueté par $\pi_{\Sigma_i}(u)$, et donc $\pi_{\Sigma_i}(u) \notin \mathcal{L}(\mathcal{A}'_i)$. Finalement, $\pi_{\Sigma_i}(u) \in \mathcal{L}(\mathcal{A}_i)$ et $\pi_{\Sigma_i}(u) \notin \mathcal{L}(\mathcal{A}'_i)$, c'est-à-dire $\mathcal{L}(\mathcal{A}_i) \neq \mathcal{L}(\mathcal{A}'_i)$. Contradiction. □

Il suffit ensuite de remarquer que, par définition, et en notant $\mathcal{A}_p^{k_p}$ l'automate $\mathcal{A}_p$ ayant pour seul état initial k_p :

$$\mathcal{L}(\langle S_i, \bigparallel_{p\in\mathcal{P}} \mathcal{A}_p\rangle) = \bigcup_{(k_1,\ldots,k_{|\mathcal{P}|})\in S_i} \mathcal{L}(\bigparallel_{p\in\mathcal{P}} \mathcal{A}_p^{k_p})$$

Ainsi, tout réseau sans blocage d'automates communicants à états finaux locaux est équivalent à une union finie de réseaux sans blocage d'automates communicants à états initiaux et

finaux locaux. Il suffit ensuite d'appliquer la proposition 2.10 pour remplacer chaque $\mathcal{A}_p^{k_p}$ par l'automate minimal et déterministe qui a le même langage.

Remarquons que cette construction ne marche pas avec la définition de réseau d'automates communicants déterministes de [Henriksen 05], ni avec les réseaux d'automates asynchrones (voir, par exemple, la figure 1.2).

Ceci conclut la seconde partie de la preuve du théorème 2.5. Nous avons donc montré que tout cHMSC régulier était équivalent (à renommage près) à une union finie de réseaux sans blocage d'automates communicants déterministes, à état initiaux et finaux locaux. Ainsi, il est possible de mettre en œuvre n'importe quel cHMSC régulier par un ensemble de programmes déterministes et parallèles, communiquant par échange de messages et dont les canaux de communication sont bornés.

2.3 Equivalence entre systèmes bornés

Dans la partie précédente, nous avons montré, avec le théorème 2.5, que les cHMSC causaux réguliers ont un langage de linéarisation régulier, et donc expriment une classe de comportements simulables par des automates. Il est, ensuite, immédiat d'appliquer tous les résultats de décidabilité issus du monde des automates aux modèles de spécification de haut niveau que nous étudions dans cette thèse.

Dans cette partie, nous allons nous intéresser à un problème d'équivalence pour une sous classe des cHMSC causaux. Nous allons montrer que les descriptions de systèmes parallèles et répartis, données à l'aide d'un réseau borné d'automates dits mixtes, car localement asynchrones et globalement communicants, sont équivalentes, au niveau de l'expressivité, aux descriptions données à l'aide de modèles de haut niveau, appelés cHMSC faiblement réguliers et cohérents.

Cette partie est organisée de la manière suivante. Tout d'abord, nous définissons dans la partie 2.3.1 la classe syntaxique des cHMSC causaux faiblement réguliers et cohérents. Puis, nous introduisons, dans la partie 2.3.2, un nouveau modèle de bas niveau composé de réseaux d'automates mixtes, car localement asynchrones et globalement communicants et nous énonçons le théorème d'équivalence pour les systèmes bornés. Enfin, ce théorème est démontré dans la partie 2.3.3.

2.3.1 HMSC causaux faiblement réguliers et cohérents

Nous allons donc montrer un résultat d'équivalence entre langages de spécifications et modèles opérationnels pour les systèmes parallèles et répartis bornés. Pour cela, nous introduisons à nouveau une restriction syntaxique des cHMSC causaux, appelés cHMSC cohérents. Cette classe va représenter des cHMSC causaux dont la relation de dépendance utilisée lors de la composition et la causalité liée aux messages permettent de définir entièrement l'ordre qui apparaît dans les cMSC causaux engendrés.

Définition 2.11 (cohérence) *Soient $(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D)$, un alphabet concurrent localisé et α, un* cHMSC *D-causal. Pour $m \in \Gamma(\alpha)$, notons $m = [E, \leq, \lambda]$, $\prec$ la réduction transitive de $\leq$, $\ll$ la causalité liée aux messages et $\prec_D = \{(e, e') \in \prec |\ \lambda(e)\ D\ \lambda(e')\}$ la causalité immédiate liée*

*à la dépendance. α est dit (D-)*cohérent *si pour tout $m \in \Gamma(\alpha)$, $\leq = (\prec_D \cup \ll)^\star$, c'est-à-dire si la causalité de chaque* MSC *causal de α est due uniquement aux échanges de messages et à la relation de dépendance.*

Le HMSC causal présenté dans la figure 2.9 est cohérent. En effet, sur chaque instance, les événements qui sont causalement reliés sont dépendants : la causalité globale peut être donc inférée uniquement à partir de D et de $\ll$.

De plus, la cohérence est une propriété syntaxique locale : elle est testable en temps polynomial en la taille du cHMSC testé. En effet, pour chaque cMSC causal fini qui apparaît dans l'expression du cHMSC causal à tester, il suffit simplement d'en calculer sa réduction transitive et d'y vérifier la condition de cohérence.

Enfin, nous donnons une définition syntaxique légèrement plus faible que la définition de la régularité que nous avons donnée dans le chapitre précédent.

Définition 2.12 (régularité faible) *Soient $(\Sigma_{\mathcal{P},\mathcal{M}\times\mathcal{D},\mathcal{I}}, D)$, un alphabet concurrent localisé, et α, c*HMSC *D-causal. α est dit* faiblement régulier *si, pour toute sous-expression $\beta^\star$ de α, pour tout $m \in \mathcal{L}_{\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I}),D)}$, le graphe orienté $(\Sigma(m), D \cup \ll)$ est fortement connexe. De plus les hypothèses (iii) et (iv) de la définition 2.4 doivent toujours être vérifiées pour assurer une politique de routage FIFO.*

Nous pouvons comparer cette définition à la régularité de la définition 2.4, pour laquelle les MSC m des boucles doivent avoir les graphes orientés $(\Sigma_p(m), D)$ et $(loc(\Sigma_p), \ll /_{\sim_{loc}})$ fortement connexes. Clairement, la définition de la régularité implique la définition de la régularité faible. De plus, cette définition donne immédiatement un algorithme CoNP-complet pour tester l'appartenance à la classes des cHMSC faiblement réguliers.

2.3.2 Réseaux d'automates mixtes

Nous présentons maintenant le modèle de bas niveau, appelé réseaux d'automates mixtes, pour lesquels nous allons montrer que les réseaux bornés correspondent exactement aux HMSC causaux faiblement réguliers et cohérents. Plus précisément, les réseaux d'automates mixtes sont des réseaux localement asynchrones et globalement communicants et, par conséquent, leur définition s'inspire directement de la définition de ces réseaux (voir les parties 1.2.3 et 1.3.4).

Définition 2.13 (Réseau d'automates mixtes) *Soit $(\mathcal{T}_p)_{p\in\mathcal{P}}$, un ensemble de noms de "threads". Un* réseau d'automates mixtes *est une structure $\mathcal{R} = \langle S_i, (\langle S_i^p, (\mathcal{A}_p^t)_{t\in\mathcal{T}_p}, S_f^p\rangle)_{p\in\mathcal{P}}, S_f\rangle$ telle que :*

- *pour tout $p \in \mathcal{P}$, $\langle S_i^p, (\mathcal{A}_p^t)_{t\in\mathcal{T}_p}, S_f^p\rangle$ est un réseau d'automates asynchrones ;*
- *le réseau $\langle S_i, (\langle S_i^p, \bigotimes_{t\in\mathcal{T}_p} \mathcal{A}_p^t, S_f^p\rangle)_{p\in\mathcal{P}}, S_f\rangle$ est un réseau d'automates communicants.*

L'automate infini qui décrit le comportement global d'un réseau d'automates mixtes est noté $\langle S_i, ||_{p\in\mathcal{P}} \langle S_i^p, \bigotimes_{t\in\mathcal{T}_p} \mathcal{A}_p^t, S_f^p\rangle, S_f\rangle$ Un réseau d'automates mixtes est dit compatible *avec D si, et seulement si, les automates $\mathcal{A}_p^t$ sont définis sur des alphabets Σ_p^t tels que les graphes (Σ_p^t, D) sont complets maximaux.*

Un automate est *D-diamant* s'il vérifie la propriété suivante : Si $q_1 \xrightarrow{a} q \xrightarrow{b} q_2$ pour deux lettres $a, b \in \Sigma$ telles que $a\ I\ b$, alors il existe q' avec $q_1 \xrightarrow{b} q' \xrightarrow{a} q_2$, avec $I = \Sigma^2 - D$. Il est facile de voir qu'un langage de traces est régulier si, et seulement si, il est reconnu par un automate D-diamant. Un réseau d'automates communicants I_p-diamants est un réseau d'automates communicants dont tous les automates $\mathcal{A}_p$ sont I_p-diamants, pour $p \in \mathcal{P}$, avec I_p la projection de I sur le processus p.

Nous montrons maintenant que la condition syntaxique de la cohérence est nécessaire et suffisante pour obtenir une équivalence entre des modèles bornés de bas niveau qui mélangent interactions par échange de messages et interactions par lectures et écritures dans une mémoire partagée et des langages de spécification décrits par des CHMSC causaux faiblement réguliers et cohérents.

Théorème 2.14 *Soient $(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D)$, un alphabet concurrent localisé, $I = \Sigma^2_{\mathcal{P},\mathcal{M},\mathcal{I}} - D$, I_p la restriction de I au processus $p \in \mathcal{P}$ et $L \subseteq \mathbb{MSC}((\mathcal{P}, \mathcal{M}, \mathcal{I}), D)$, un ensemble de* MSC. *Alors, les propositions suivantes sont équivalentes :*

- *L est le langage d'un* CHMSC *D-causal faiblement régulier et D-cohérent ;*
- *L est le langage (avec ajout de données) d'un réseau sans blocage et universellement borné d'automates communicants I_p-diamants ;*
- *L est le langage (avec ajout de données) d'un réseau sans blocage et universellement borné d'automates mixtes à états finaux et compatibles avec D.*

2.3.3 Preuve du Théorème 2.14 (équivalence de modèles)

Dans cette partie, nous allons démontrer le théorème 2.14, à savoir l'équivalence (à renommage près) entre spécifications écrites en CHMSC causaux réguliers et cohérents (lorsque la relation d'indépendance est respectée à l'intérieur des MSC causaux) et un modèle opérationnel représenté par des réseaux sans blocage et universellement bornés d'automates mixtes (qui sont localement asynchrones et globalement communicants).

Pour cela, nous allons nous appuyer sur plusieurs lemmes, dont l'organisation est résumée dans la figure 2.11.

Une partie des preuves que nous fournissons dans cette partie s'appuie sur le fait que le réseau d'automates communicants considéré est sans blocage. Cette hypothèse primordiale ne nous permet pas, malheureusement, d'étendre cette construction au cas des CHMSC causaux globalement coopératifs et des réseaux existentiellement bornés d'automates mixtes. En effet, il n'existe pas (et il semble difficile que cela soit possible) de caractérisation des langages de CHMSC globalement coopératifs en terme de réseau d'automates communicants sans blocage.

Lemme 2.15 *Soient $(\Sigma_{\mathcal{P},\mathcal{M}\times\mathcal{D},\mathcal{I}}, D)$, un alphabet concurrent localisé et $L \subseteq \Sigma^\star_{\mathcal{P},\mathcal{M}\times\mathcal{D},\mathcal{I}}$ un langage régulier. L est le langage d'un réseau sans blocage et universellement borné d'automates communicants D_p-diamants et à états finaux locaux si, et seulement si, L est le langage d'un réseau sans blocage et universellement borné d'automates mixtes compatibles avec D et à états finaux locaux.*

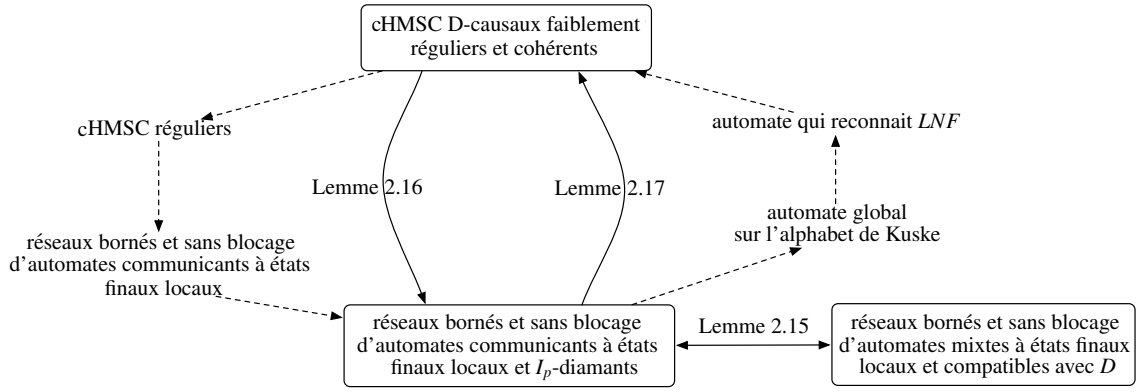


FIG. *2.11 – Résumé des étapes de la preuve d'équivalence.*

Preuve: Soit $\langle S_i, (\mathcal{A}_p)_{p\in\mathcal{P}}\rangle$, un réseau sans blocage et universellement borné d'automates communicants D_p-diamants et à états finaux locaux. Par définition du langage d'un réseau d'automates communicants, ce réseau est équivalent à une union de réseaux $(\mathcal{A}_p^k)_{p\in\mathcal{P}}$ sans blocage et universellement bornés d'automates communicants à états initiaux et finaux locaux. De plus, le langage est toujours régulier, ces réseaux sont donc universellement bornés ([Henriksen 05]). Ainsi, en notant $\mathcal{A}_p^{k_p}$ la copie de $\mathcal{A}_p$ ayant k_p comme seul état initial, nous avons :

$$\mathcal{L}(\langle S_i, \mathop{||}_{p\in\mathcal{P}} \mathcal{A}_p\rangle) = \bigcup_{(k_1,\ldots,k_{|\mathcal{P}|})\in S_i} \mathcal{L}(\mathop{||}_{p\in\mathcal{P}} \mathcal{A}_p^{k_p})$$

$\mathcal{A}_p$ étant D_p-diamant, chaque $\mathcal{A}_p^k$ est D_p-diamant. Nous pouvons alors utiliser la proposition 2.10 et le résultat de [Zielonka 89, Baudru 07] (th. 1.14) pour transformer chaque automate local en réseau sans blocage d'automates asynchrones à états finaux locaux. Ainsi, chaque $\mathcal{A}_p^{k_p}$ peut être décomposé en un ensemble de "threads" $\mathcal{A}_{p,t}^{k_p}$, pour $t \in \mathcal{T}_p$. Plus précisément :

$$\mathcal{L}(\langle S_i, \mathop{||}_{p\in\mathcal{P}} \mathcal{A}_p\rangle) = \bigcup_{(k_1,\ldots,k_{|\mathcal{P}|})\in S_i} \mathcal{L}(\mathop{||}_{p\in\mathcal{P}} \langle S_i^p, \bigotimes_{t\in\mathcal{T}_p} \mathcal{A}_{p,t}^{k_p}\rangle)$$

Enfin, par définition, nous avons : $\mathcal{L}(\langle S_i, ||_{p\in\mathcal{P}} \mathcal{A}_p\rangle) = \mathcal{L}(\langle S_i, ||_{p\in\mathcal{P}} \langle S_i^p, \bigotimes_{t\in\mathcal{T}_p} \mathcal{A}_{p,t}\rangle\rangle)$.

Inversement, étant donné un réseau sans blocage et universellement borné d'automates mixtes compatibles avec D et à états finaux locaux, il est possible, en appliquant la proposition 2.10, de construire une union de réseaux universellement bornés d'automates communicants sans blocage et à états initiaux et finaux locaux, dont le langage de chaque automate local est D_p-diamant. Cette union de réseaux peut ensuite être représentée par un unique réseau, en choisissant des états initiaux globaux adaptés. Ce qui conclut la preuve de ce lemme. □

Lemme 2.16 *Soient $(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D)$, un alphabet concurrent localisé, $I = \Sigma^2_{\mathcal{P},\mathcal{M},\mathcal{I}} - D$, I_p, la projection de I sur le processus p et α, un* cHMSC *causal D-cohérent et régulier. Alors, α peut être mis en œuvre avec données par un réseau sans blocage et universellement borné d'automates communicants à états finaux locaux et I_p-diamants.*

Preuve: Nous allons montrer ce résultat en trois étapes. La première étape consiste à traduire un cHMSC causal faiblement régulier et cohérent en cHMSC régulier. Notons qu'en l'absence de l'hypothèse de cohérence dans la partie 2.2.3, nous avons du faire une preuve beaucoup plus technique pour montrer un résultat similaire. La seconde étape consiste à utiliser [Baudru 07] (théorème. 1.32) pour obtenir un réseau sans blocage et borné d'automates communicants à états finaux. Enfin, la dernière étape consiste à faire localement le produit de ces automates pour obtenir le résultat désiré et à montrer que les automates communicants $\mathcal{A}_p$ sont bien I_p-diamants.

Construction d'un cHMSC à partir d'un cHMSC causal Etant donné $\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$ un alphabet global étiquetant un cMSC causal, notons $\Sigma_{\widehat{\mathcal{P}},\widehat{\mathcal{M}},\widehat{\mathcal{I}}}$ l'alphabet global défini sur l'alphabet de processus $\widehat{\mathcal{P}} = 2^{\mathcal{P}}$, sur l'alphabet de messages $\widehat{\mathcal{M}} = (\mathcal{M} \cup \{m_a \mid a \in \Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}\})$ et l'alphabet d'actions internes $\widehat{\mathcal{I}} = \Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$. Soit $a \in \Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$, une action. L'ensemble des threads associés à a, noté $thr(a)$ est l'ensemble des $T \in 2^{\mathcal{P}}$ maximaux tels que (T, D) forme un graphe complet et tels que $a \in T$.

Pour cette preuve, nous devons considérer l'état des canaux, donc nous reprenons la notation de la partie 1.3.3. Fixons (c, χ), un cMSC D-causal, avec c un pomset étiqueté par $\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$ et χ un état de canal.

Considérons, tout d'abord, un événement e de E_c. Notons $a = \lambda(e)$ et notons t_k le "plus grand" (pour l'ordre lexicographique par exemple) thread de $thr(a) = \{t_1 \dots t_k\}$. Nous avons alors :

- si $a = p(i)$, alors notons $\widehat{e}$ un événement étiqueté par $t_k(i)$;
- si $a = p!q(m)$ et u_l est le plus grand tread de $thr(q?p(a))$, alors notons $\widehat{e}$ un événement étiqueté par $t_k!u_l(m)$;
- si $a = p?q(m)$ et u_l est le plus grand tread de $thr(q!p(a))$, alors notons $\widehat{e}$ un événement étiqueté par $t_k?u_l(m)$.

Nous construisons, ensuite, le pomset $\widehat{c}_e$ étiqueté par $\Sigma_{\widehat{\mathcal{P}},\widehat{\mathcal{M}},\widehat{\mathcal{I}}}$, constitué d'une chaîne de message de type m_a, allant de t_1 à t_k. Cette barrière est suivie de l'action $\widehat{e}$ sur le processus t_k et enfin, d'une seconde chaîne de messages de type m_a allant, cette fois-ci de t_k à t_1. La figure 2.12 illustre cette construction, avec $\lambda(e) = p(i)$ et $thr(p(i)) = \{t_1, t_2, t_3\}$.

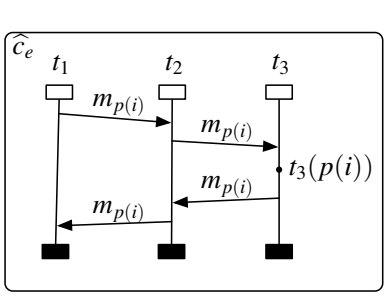


FIG. *2.12 – $\widehat{c}_e$, pour un événement e de E_m tel que $\lambda(e) = p(i)$ et $thr(p(i)) = \{t_1, t_2, t_3\}$.*

Etant donné (c_1, χ_1) et (c_2, χ_2) tels que $(c, \chi) = (c_1, \chi_1) \circ (c_2, \chi_2)$ et e un élément minimal de c_2, il est facile de caractériser $\widehat{\chi}_e$, l'état du canal juste avant l'exécution de $\widehat{c}_e$: il suffit d'associer le contenu du canal (p, m, q) de χ_2 au canal (t, m, u) de $\widehat{\chi}_e$, où t et u sont les plus

grands threads de $thr(p)$ et $thr(q)$ et d'associer 0 à tous les nouveaux canaux créés. Etant donné un événement e du cMSC causal (c, χ), on peut donc construire un cMSC $(\widehat{c}_e, \widehat{\chi}_e)$ correspondant.

Enfin, cette construction peut se généraliser à tout cMSC causal (c, χ) : soit $e_1 < \ldots < e_{|E_c|}$ une extension linéaire de c qui définit une suite de cMSC causaux $(c_i, \chi_i) = (\{e_i\}, \chi_{e_i})$ avec χ_{e_i} construit à partir de la décomposition de (c, χ) en $(c_1, \chi_1) \circ \ldots \circ (c_{i-1}, \chi_{i-1})$ et $(c_i, \chi_i) \circ \ldots \circ (c_{|E_c|}, \chi_{|E_c|})$. Alors, nous pouvons construire $(\widehat{c}, \widehat{\chi}) = (\widehat{c}_{e_1}, \widehat{\chi}_{e_1}) \circ \ldots \circ (\widehat{c}_{e_{|E_c|}}, \widehat{\chi}_{e_{|E_c|}})$, le cMSC correspondant au cMSC causal (c, χ).

Enfin, définissons $\phi : \Sigma_{\widehat{\mathcal{P}},\widehat{\mathcal{M}},\widehat{\mathcal{I}}} \to \Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$ comme la fonction suivante : $\phi(t(a)) = a$ et sinon $\phi(b) = \varepsilon$. Cette fonction peut être étendue aux ensembles étiquetés $\phi((E, \lambda)) = (\{e \mid \phi(e) \neq \varepsilon\}, \phi)$ puis aux pomsets : $\phi([E, \leq, \lambda]) = (\phi(E), \leq \cap \phi(E)^2, \lambda])$. Remarquons clairement que si c est *cohérent*, alors $\phi(\widehat{c})$ est exactement c.

Etant donné un cHMSC causal α, notons $\widehat{\alpha}$ le cHMSC obtenu en remplaçant chaque cMSC causal (c, χ) apparaissant dans l'expression rationnelle par le cHMSC $(\widehat{c}, \widehat{\chi})$ construit par la méthode définie ci-dessus. Là encore, remarquons que si α est *cohérent*, l'ensemble des pomsets étiquetés par $\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$ engendré par $\phi(\mathcal{L}_{\mathbb{MSC}(\widehat{\mathcal{P}},\widehat{\mathcal{M}},\widehat{\mathcal{I}})}(\widehat{\alpha}))$ est exactement $\mathcal{L}_{(\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\alpha)$.

Mise en œuvre du cHMSC Remarquons ensuite que si α est faiblement régulier, alors $\widehat{\alpha}$ est un HMSC régulier. Supposons maintenant que nous avons un cHMSC causal α *cohérent* et *faiblement régulier*. En appliquant directement [Baudru 07] (théorème 1.32) sur le cHMSC $\widehat{\alpha}$, on obtient un réseau $\mathcal{C}(\widehat{\alpha})$ universellement borné et sans blocage d'automates communicants à états finaux locaux qui met en œuvre $\widehat{\alpha}$, avec ajout de données. On obtient alors que $\phi(\mathcal{L}(\mathcal{C}(\widehat{\alpha})))$ correspond exactement à $\phi(Lin(\widehat{\alpha})) = Lin(\alpha)$, modulo des ajouts de données.

Produit des automates communicants locaux A partir de $\mathcal{C}(\widehat{\alpha})$, nous cherchons maintenant à reconstruire un réseau d'automates communicants $\mathcal{C}'$ qui travaille sur les processus $\mathcal{P}$ (et non plus sur $\widehat{\mathcal{P}} = 2^{\mathcal{P}}$). Pour cela, pour toutes les lettres $a \in \Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$ telles que $loc(a) = \{t_1 \ldots t_r\}$, considérons les automates $\mathcal{A}_{t_1} \ldots \mathcal{A}_{t_r}$ de $\mathcal{C}(\widehat{\alpha})$ correspondants. Si il existe des états x_i, n_i, n'_i, n''_i et y_i de $\mathcal{A}_{t_i}$ pour $1 \leq i \leq r$ et des données $d_1 \ldots d_{2r-2}$ tels que :

- $x_1 \xrightarrow{t_1!t_2(m_a,d_1)} n_1 \xrightarrow{t_1?t_2(m_a,d_{2r-2})} y_1$;
- $x_r \xrightarrow{t_r?t_{r-1}(m_a,d_{r-1})} n_r \xrightarrow{\widehat{a}} n'_r \xrightarrow{t_r!t_{r-1}(m_a,d_r)} y_1$, $\widehat{a}$ ayant comme forme :
  - $t_r(p(i))$ si a est une action interne, c'est à dire si $a = p(i)$;
  - $t_r!u(m, d)$ si a est un envoi de message, c'est à dire si $a = p!q(m)$;
  - $t_r?u(m, d)$ si a est une réception de message, c'est à dire si $a = p?q(m)$.
- $x_i \xrightarrow{t_i?t_{i-1}(m_a,d_{i-1})} n_i \xrightarrow{t_i!t_{i+1}(m_a,d_i)} n'_i \xrightarrow{t_i?t_{i+1}(m_a,d_{2r-i-1})} n''_i \xrightarrow{t_i!t_{i-1}(m_a,d_{2r-i})} y_i$.

Alors, nous ajoutons les états $x = (x_1, \ldots, x_r)$ et $y = (y_1, \ldots, y_r)$ à l'automate $\mathcal{A}_{loc(a)}$ de $\mathcal{C}'$, ainsi qu'une transition $x \xrightarrow{a'} y$, où $a' = a$ si a est une action interne, mais $a' = p!q(m, d)$ ou $a' = p?q(m, d)$ si a correspond à un envoi ou une réception de données (avec d la donnée rajoutée par $\mathcal{A}_{t_r}$).

De plus, étant donné un état $z = (z_1, \ldots, z_k)$ de $\mathcal{C}'$, si pour tous les i tels que $1 \leq i \leq k$, z_i est final dans $\mathcal{A}_{t_i}$, alors nous ajoutons z aux états finaux de $\mathcal{A}_{loc(a)}$. De plus, si $(n_1, \ldots, n_{|2^{\mathcal{P}}|})$

est un état initial de $\mathcal{C}(\widehat{\alpha})$, alors nous ajoutons, pour chaque $T = \{t_1 \dots t_k\} \subseteq 2^{\mathcal{P}}$ tel que $\exists p \in \mathcal{P}$ avec $p \in t_i$, $(n_{t_1}, \dots, n_{t_k})$ aux états initiaux de $\mathcal{C}(\widehat{\alpha})$.

Il est alors immédiat de voir que les $\mathcal{A}_p$ ainsi définis sont I_p diamants : en effet, si ab peut être tiré dans $\mathcal{A}_p$, avec $a\, I_p\, b$, alors notons $thr(a) = \{t_1 \dots t_k\}$ et $thr(b) = \{t_{k+1} \dots t_l\}$. Comme a et b sont indépendants, tous les t_i sont distincts, pour $1 \leq i \leq i$. Il existe donc des états $x_1 \dots x_l$ et $y_1 \dots y_l$ dans les automates $\mathcal{A}_{t_1} \dots \mathcal{A}_{t_l}$ de $\mathcal{C}(\widehat{\alpha})$, tels que $q_1 = (\dots, x_1 \dots x_l \dots) \xrightarrow{a} (\dots y_1 \dots y_k x_{k+1} \dots x_l) \xrightarrow{b} q_2 = (\dots y_1 \dots y_k y_{k+1} \dots y_l)$ dans $\mathcal{A}_p$. Nous pouvons donc trouver un état $q' = (\dots x_1 \dots x_k y_{k+1} \dots y_l)$ tel que $q_1 \xrightarrow{b} q' \xrightarrow{a} q_2$, ce qui montre finalement que $\mathcal{A}_p$ est I_p-diamant.

Enfin, par construction, le nouveau réseau $\mathcal{C}'$ est un réseau borné et sans blocage d'automates communicants à états finaux dont le langage est, modulo ajout de données, exactement $\phi(Lin(\widehat{\alpha}))$. Il met donc en œuvre, avec ajout de données, le cHMSC causal $Lin(\alpha)$. □

Lemme 2.17 *Soient $(\Sigma_{\mathcal{P},\mathcal{M}\times\mathcal{D},\mathcal{I}}, D)$, un alphabet concurrent localisé, $I = \Sigma^2_{\mathcal{P},\mathcal{M},\mathcal{I}} - D$, I_p, la projection de I sur le processus p et $\mathcal{C} = \langle S_i, (\mathcal{A}_p)_{p\in\mathcal{P}}\rangle$, un réseau sans blocage et universellement borné d'automates communicants à états finaux locaux et I_p-diamant. Alors, il existe α, un cHMSC D-causal faiblement régulier et cohérent que $\mathcal{C}$ met en œuvre.*

Preuve: La preuve s'effectue en trois étapes, de manière quasiment identique à ce qui a été proposé par Genest dans [Genest 05], page 80.

La première étape consiste à calculer la borne b du réseau $\mathcal{C}$. Grâce à cette borne b, il est possible d'encoder, dans les actions, l'état du canal, à la manière de Kuske dans [Kuske 03]. On peut ainsi, dans une deuxième étape, remplacer les actions $a!q(m,d)$ ou $a = p?q(m,d)$ par des actions (a,i) avec $0 \leq i \leq b-1$, où i est le nombre, modulo b, de messages déjà envoyés entre p et q au moment où l'événement étiqueté par a a lieu. De même, on remplace les actions $a = p(i)$ par $(a,0)$. Il faut alors adapter l'encodage original de Kuske pour introduire une nouvelle relation de dépendance $\widehat{D}$, qui prend en compte la relation de dépendance D définie sur les actions. Cette nouvelle relation est définie de la manière suivante : $(a,i)\,\widehat{D}\,(b,j)$ si $a\,D\,b$ ou si $a = p!q(m,d)$, $b = q?p(m,d)$ et $j = i+1 \bmod b$ (et inversement). On note alors $((\Sigma_{\mathcal{P},\mathcal{M},\mathcal{D}}, b), \widehat{D})$ l'alphabet concurrent qui correspond à ce nouvel étiquetage et $\widehat{\mathcal{C}}$ le réseau d'automates communicants obtenu par cette transformation.

La troisième étape consiste à calculer l'automate global $\mathcal{A}_g$, à partir du réseau d'automates communicants $\widehat{\mathcal{C}}$. Il est clair que $\mathcal{A}_g$ est $\widehat{I}$-diamant, avec $\widehat{I} = (\Sigma_{\mathcal{P},\mathcal{M},\mathcal{D}}, b)^2 - \widehat{D}$ et que l'image du langage de $\mathcal{A}_g$ par $\theta((a,i)) = a$ donne le même langage que celui de $\mathcal{C}$.

Pour la dernière étape, Ochmanski a montré dans [Ochmanski 85] qu'il était possible de construire un $\Sigma^\star_{\mathcal{P},\mathcal{M},\mathcal{I}}$-automate $\mathcal{A}_{LNF}$ potentiellement plus gros qui reconnaisse un ensemble de représentants du langage de traces de $\mathcal{A}_g$ ($\mathcal{A}_g$, qui lui est clos par commutation). Plus précisément, ces représentants sont les *formes normales lexicographiques* du langage de traces $\mathcal{L}_{\mathbb{M}(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}},D)}(\mathcal{A}_g)$ définis comme suit : un mot $v \in \Sigma^\star_{\mathcal{P},\mathcal{M},\mathcal{I}}$, v est la forme normale lexicographique de la trace w si, et seulement si, pour toute décomposition $v = xabz$ avec aIb, $a < b$, avec $<$ un ordre total sur $\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}$. Ochmanski a, de plus, montré que cet automate

avait des boucles D-connexes. Il suffit alors de prendre l'image de $\mathcal{A}_{LNF}$ par $\phi((a,i)) = a$ et d'interpréter le résultat comme un cHMSC D-causal pour obtenir le résultat voulu. Notons, en particulier, que ce résultat implique que le cHMSC reconstruit est bien faiblement régulier mais qu'il a peu de chance d'être régulier. □

2.4 Conclusions et perspectives

Résumé

Dans ce chapitre, nous avons commencé à étendre aux cHMSC causaux, les nombreux résultats concernant la mise en œuvre de cHMSC par des réseaux d'automates communicants. En particulier, nous avons montré deux résultats importants.

D'une part, nous avons montré que la classe des cHMSC causaux réguliers peut être simulée par des automates. Cette traduction permet donc d'utiliser les nombreuses propriétés décidables de ces structures (en particulier : décidabilité des problèmes de l'appartenance, de l'inclusion et de la vacuité de l'intersection) afin de proposer des solutions à des problèmes réputés difficiles pour les systèmes parallèles et répartis. Cependant, bien que le pouvoir d'expression des HMSC causaux réguliers soit plus grand que celui des HMSC réguliers (car ils permettent d'exprimer des comportements qui ne sont pas finiment engendrés), le pouvoir d'expression des cHMSC causaux réguliers est identique à celui des cHMSC réguliers. En particulier, les cHMSC causaux réguliers permettent de spécifier seulement une classe restreinte de systèmes parallèles et répartis, les systèmes dont les canaux de communication ont une capacité bornée. Afin de résoudre ce problème d'expressivité, nous verrons, dans le chapitre suivant, comment généraliser une partie de bonnes propriétés de décision liées aux automates, à des systèmes infinis.

D'autre part, nous avons montré que la classe des cHMSC causaux réguliers et cohérents a un pouvoir d'expression équivalent à celui des réseaux sans blocage et universellement bornés d'automates mixtes, c'est-à-dire des automates localement asynchrones et globalement coopératifs. Ce résultat permet de caractériser une sous-classe intéressante (bien qu'ayant un espace d'états bornés) de réseaux parallèles et répartis complexes, mélangeant des communications par mémoire partagée et par envois de messages asynchrones. Cependant, encore une fois, ces résultats d'équivalence se restreignent aux systèmes bornés. Nous n'avons pas de solution, pour le moment, afin d'obtenir des résultats similaires sur des systèmes non bornés, principalement car nous ne savons pas caractériser les cHMSC globalement coopératifs qui peuvent être mis en œuvre sans blocage.

Perspectives

De nombreux problèmes évoqués dans cette partie restent ouverts. En effet, l'état de l'art présenté en début de ce chapitre indique qu'il existe plusieurs techniques de mise en œuvre et nous nous sommes intéressés, dans cette thèse, uniquement à celles qui ajoutent des données dans les messages et qui manipulent des systèmes bornés. Ainsi, nous pouvons fixer, pour continuer les travaux entamés dans ce chapitre, deux perspectives principales.

La première perspective consisterait à généraliser les résultats obtenus sur les cHMSC pour la mise en œuvre sans ajout de données. La classe "idéale" à étendre est, dans ce cas, celle des cHMSC à choix local qui permettent de répartir le contrôle global sur les différents processus. Les cHMSC qui sont dans cette classe définissent des comportements où tous les processus impliqués dans un choix global possèdent, dans leur passé causal, un événement commun. Le processus sur lequel est localisé cet événement est chargé d'effectuer à l'avance ce choix global. En fonction de l'étiquette qu'il donne à cet événement commun, chaque processus impliqué dans le choix est capable d'effectuer localement un choix cohérent avec ceux des autres processus. Dans le cadre des cHMSC causaux, la démarche resterait la même, la modification à apporter entrerait dans le calcul du passé causal afin de trouver un événement commun.

La première solution consisterait à simuler exactement l'ensemble des MSC que peut générer un cHMSC considéré. Le passé considéré est celui de tous les ordres causaux générés. Cependant, avec cette définition, plus la relation d'indépendance I est forte, plus la mise en œuvre sans donnée est difficile : en effet, augmenter I va diminuer le nombre de chaînes causales dans le passé de chaque processus, et il sera plus difficile d'y trouver un événement unique situé sur le processus chargé d'effectuer le choix global.

La seconde solution consisterait à profiter de la relation d'indépendance pour choisir, pour chaque ordre causal généré par le HMSC causal considéré, un unique MSC à simuler. Ainsi, augmenter la relation d'indépendance simplifierait la mise en œuvre. Une mise en œuvre donnée correspondrait alors à un choix, effectué par un ordonnanceur de "threads" attaché à chaque processus, de l'exécution qui facilite le plus la mise en œuvre de la spécification globale. Un HMSC causal serait alors vu comme une spécification pour laquelle une certaine liberté est pour la mise en œuvre.

La seconde perspective consiste à généraliser les résultats obtenus sur les cHMSC causaux réguliers pour la mise en œuvre, avec données, de comportements bornés. En effet, la critique principale que nous pouvons faire sur nos travaux est le fait que nous obtenons des résultats d'équivalence entre modèles de spécification et modèles d'exécution uniquement pour des systèmes bornés. Or, en général, cette borne dépend du réseau sous-jacent à l'exécution de notre système, et c'est un paramètre sur lequel nous ne pouvons influer. C'est même un paramètre qui, en général, n'est pas connu. Dans ce cadre, l'intérêt de ce chapitre est de donner les techniques de base qui permettront d'aller vers des équivalences de systèmes non bornés. Les résultats obtenus utilisent principalement le fait que les cHMSC réguliers peuvent être mis en œuvre sans blocage. Afin de généraliser nos résultats aux systèmes infinis, il faudrait donc être capable de répondre à la question : un cHMSC globalement coopératif peut-il être mis en œuvre avec données par un réseau sans blocage d'automates communicants ? Bien qu'il semble difficile d'avoir une caractérisation syntaxique d'une classe de tels cHMSC, la proposition 2.10 peut, peut-être, nous permettre de définir un test de mise en œuvre sans blocage. En nous appuyant sur ce résultat, il nous semble possible, ensuite, d'utiliser les mêmes techniques de preuve pour identifier une sous-classe des cHMSC causaux globalement coopératifs et une sous-classe des réseaux (non bornés) d'automates communicants. Dans une perspective à long terme, identifier clairement de telles sous-classes nous semble essentiel.

Deuxième partie

Vérification

Chapitre 3

Vérification de modèles

Nous avons vu, dans la partie I de ce document, différents modèles pour spécifier des systèmes parallèles et répartis et nous avons introduit un modèle mixte, localement asynchrone et globalement communicant qui les généralise.

Ce chapitre constitue, avec le chapitre suivant, la partie II de cette thèse, consacrée à la *vérification* de systèmes mixtes. Plus précisément, dans ce chapitre, nous nous intéressons à la *vérification complète* de modèles. Ce qui signifie que nous étudions des techniques qui ont vocation à se dérouler avant l'exécution du système à vérifier. De plus, nous supposons que les propriétés à tester portent sur l'ensemble du modèle à vérifier.

Plus précisément, nous proposons, dans ce chapitre, deux nouvelles approches pour la vérification de spécifications de systèmes parallèles et répartis.

Dans la première approche, nous étendons les logiques temporelles classiques aux ordres partiels. Nous considérons, ensuite, que la spécification à tester est écrite dans un langage de haut niveau et décrit, de manière concise en utilisant elle-aussi des ordres partiels, le schéma d'interaction des différentes entités du système. Nous montrons, dans ce cadre, qu'il existe des méthodes efficaces de "model-checking", ayant des complexités similaires à celles de la vérification de systèmes séquentiels. Ce résultat démontre l'intérêt d'utiliser des ordres partiels pour modéliser des systèmes parallèles et répartis : nous gagnons en concision grâce à la modélisation, sans rien perdre au niveau de l'efficacité de la vérification : la perte se fait au niveau de l'expressivité des modèles et des logiques.

Dans la seconde approche, nous généralisons le langage de spécification utilisé pour décrire le modèle, mais nous restreignons la puissance des propriétés vérifiables. Nous montrons que, là encore, les modèles d'ordres partiels permettent de vérifier une large classe de systèmes que les méthodes classiques de vérification ne sont pas capables de réaliser.

Le chapitre suivant, lui aussi consacré à la vérification, se focalisera sur des analyses partielles de modèles. Enfin, la partie III de ce document sera consacrée à la supervision de systèmes parallèles et répartis mixtes.

Contexte

Les programmes informatiques sont de plus en plus gros et complexes. Malheureusement, les méthodes manuelles classiques de validation de logiciels (revues de code, simulations, tests) ne passent pas à l'échelle. La production de logiciels fiables, leur maintenance et leur évolution au fil du temps doivent donc passer par une automatisation : cela revient à demander à l'ordinateur de trouver lui-même les erreurs commises par le programmeur. Cet ensemble de techniques automatiques, qui consistent à vérifier un certain nombre de propriétés de bon fonctionnement d'un programme avant même son exécution, forme le domaine de la *vérification statique de programmes*.

Logiques temporelles

Le formalisme des logiques temporelles permet de raisonner à propos des propriétés d'un système, liées à son évolution au cours du temps. Ce formalisme existe dans de nombreux domaines comme la physique, la philosophie, la linguistique, et il est aussi très utilisé en informatique théorique, et plus particulièrement en logique. En particulier, ce formalisme a été introduit par Pnueli dans le domaine de la vérification de programmes ([Pnueli 77, Pnueli 81]) où, il a rapidement été montré qu'il permettait de décrire aussi bien des propriétés de vivacité, exprimant le fait qu'une bonne propriété peut arriver au système ([Owicki 82]) ou de sûreté, exprimant le fait qu'une mauvaise propriété n'arrivera jamais au système ([Francez 86]).

Il existe deux types de logiques temporelles : celles qui parlent de "temps linéaire" et celles qui parlent de "temps branchant" ([Emerson 90]). D'une part, dans les logiques temporelles linéaires, on ne considère pour chaque instant qu'un seul futur possible, alors que, d'autre part, dans les logiques temporelles branchantes, chaque instant dans le temps en a plusieurs. De plus, certaines logiques permettent de spécifier des propriétés sur les états et les actions produites par un modèle, d'autres ne permettent d'exprimer que des propriétés sur les actions produites. La figure 3.1 montre un classement partiel des nombreuses logiques temporelles existantes, en fonction de ces paramètres.

	états / actions	uniquement actions
temps linéaire	LTL	HML [Hennessy 85]
temps branchant	CTL, CTL*	ACTL, ACTL* [Nicola 90] PDL [Fischer 79]

FIG. *3.1 – Un aperçu des différentes logiques temporelles.*

Model-checking

Le "model-checking" désigne les techniques et algorithmes mis en jeu afin de vérifier qu'une structure donnée (appelée le *modèle*) est compatible avec un ensemble de propriétés. Ce concept est très général, mais dans les faits, le modèle et les propriétés appartiennent à des familles spécifiques de structures.

D'une part, le modèle est, en général, représenté par une structure contenant l'ensemble des états possibles de la mémoire associée à l'exécution d'un programme. Ainsi, chaque état affecte une valeur à toutes les variables utilisées et mémorise le point courant de l'exécution

du programme. Les programmes correspondant à ces structures possèdent des variables dont la valeur appartient à un domaine fini et possèdent des appels récursifs bornés. Bien que restrictive, cette classe de modèles permet néanmoins de décrire nombre de protocoles de communication (voir [Liu 89] par exemple). Chaque état d'un programme fini peut être décrit de manière finie, et donc, par conséquent, être décrit par un nombre fini de propositions booléennes atomiques. Ce type de structure logique est appelé *structure de Kripke*.

D'autre part, les propriétés à vérifier sur le modèle peuvent être exprimées à l'aide de formules d'une logique temporelle quelconque qui utilise les propositions atomiques du modèle.

Le problème du model-checking revient alors à vérifier automatiquement que la structure de Kripke, correspondant à un programme, satisfait une formule logique exprimée dans une logique temporelle donnée. Cette technique est donc automatique, contrairement aux méthodes de preuves semi-automatiques, qui demandent l'intervention d'un être humain utilisant l'ordinateur afin de guider la preuve de satisfiabilité. Pour plus de détails concernant le model-checking, se rapporter à [Kurshan 94, Holzmann 97, Clarke 99, Berard 01].

Model-checking fondé sur les automates

Dans ce chapitre, nous nous concentrons sur une approche fondée sur la théorie des automates afin de résoudre le problème du model-checking.

Pour les deux types de logique (linéaires et branchantes), des techniques algorithmiques fondées sur les automates ont été développées avec succès, permettant d'obtenir les mêmes complexités que les techniques logiques classiques utilisées pour ces modèles. L'idée de base est d'associer un automate d'une classe particulière à chaque propriété à vérifier, et ensuite, d'utiliser des techniques classiques d'algorithmique sur les automates (comme l'intersection, le test de vacuité, etc.) afin de comparer cet automate et la structure de Kripke correspondant au programme.

Pour les logiques linéaires, les automates correspondants sont les automates de mots infinis [Lichtenstein 85b, Sistla 87, Vardi 94b], alors que pour les logiques branchantes, ce sont les automates d'arbres infinis [Emerson 84, Streett 84, Emerson 85, Emerson 88, Vardi 86b]. Les premières traductions construisaient des automates non déterministes [Vardi 86b, Vardi 94b] et avaient un coût exponentiel. Au début des années 1990, une autre traduction, vers des automates alternants, a été proposée dans [Muller 88, Kupferman 00, Vardi 94a], afin de retrouver les complexités données par les approches logiques. Nous ne détaillons pas ici le fonctionnement de tels automates (se reporter par exemple à [Muller 88]), mais le point principal est que la traduction d'une formule logique est un automate alternant, dont la taille est équivalente à un facteur multiplicatif près.

Abstraction

En pratique, le model-checking a besoin, pour fournir une analyse exacte, d'une représentation finie de la spécification à analyser. Ceci oblige soit, à réduire la précision de la spécification en l'abstrayant, et donc la précision du résultat de l'analyse, soit, à effectuer une vérification partielle. Dans tous les cas, le facteur limitant est lié à une explosion des états accessibles par la spécification à analyser. Plusieurs familles de techniques permettent cependant de limiter ce phénomène, telles que les réductions d'ordres partiels ou le model-checking symbolique

(voir, par exemple, [Clarke 99] ou [Berard 01]).

Pour pallier ces difficultés dans le cadre de la vérification automatique de spécifications de systèmes parallèles et répartis mixtes, nous proposons deux nouvelles approches pour le model-checking de spécifications de systèmes parallèles et répartis. Ces deux approches adaptent les techniques classiques de model-checking aux modèles d'ordres partiels étiquetés et elles consistent soit, à limiter l'expressivité du formalisme utilisé pour spécifier les modèles, soit, à limiter l'expressivité du formalisme utilisé pour spécifier les formules.

Organisation du chapitre

Ce chapitre est organisé de la manière suivante. Tout d'abord, la partie 3.1 présente un bref état de l'art sur la théorie du model-checking. Nous donnons dans cette partie les résultats essentiels, que nous utiliserons dans la suite de ce chapitre pour prouver les complexités des différentes solutions proposées.

Ensuite, nous présentons, dans la partie 3.2, nos résultats concernant le model-checking de spécifications décrites à l'aide de générateurs de familles de pomsets. Ces résultats sont, à nos yeux, des résultats importants, car ils motivent l'utilisation d'ordres partiels pour la modélisation et l'analyse de systèmes parallèles et répartis : en effet, nous montrerons que nous gagnons sur la taille des modèles sans pour autant perdre d'efficacité lors de l'analyse.

Puis, dans la partie 3.3, nous appliquons directement ces résultats aux HMSC causaux. Ceci ouvre le champ de l'analyse de systèmes parallèles et répartis mixtes, composés à la fois d'entités communiquant par mémoire partagée et par échange de messages asynchrones.

Enfin, la partie 3.4 conclut ce chapitre en donnant quelques perspectives. En particulier, nous discutons de la portée des résultats énoncés dans ce chapitre.

3.1 Etat de l'art

Dans la partie suivante, nous définissons trois logiques très utilisées pour les systèmes séquentiels, puis nous donnons quelques résultats concernant les logiques d'ordres partiels.

LTL,CTL et CTL*

Dans cette partie, nous définissons les trois logiques temporelles classiques des systèmes séquentiels. Ces logiques sont très utilisées, car la complexité du model-checking pour les formules de ces logiques reste très raisonnable et applicable en pratique. Les définitions qui vont suivre sont fortement inspirées de [Schnoebelen 02] et du chapitre de Vardi dans [Blackburn 06]. Pour commencer, nous fixons pour l'ensemble de cette partie, un ensemble de propositions atomiques $\Sigma = \{a_1, a_2, \ldots\}$. Nous commençons par définir les structures de Kripke, structures sur lesquelles nous allons interpréter les formules de LTL, CTL ou CTL*, puis nous détaillons les différentes logiques temporelles que nous allons étudier dans ce chapitre.

Structures de Kripke et automates.

Une *structure de Kripke* sur Σ est une structure finie $\langle Q, R, \lambda, Q_i\rangle$ où $Q = \{q_1, q_2, \ldots\}$ est un ensemble d'états, $R \subseteq Q \times Q$ est une relation de transitions, $\lambda : Q \to 2^\Sigma$ est une fonction

qui étiquette chaque état par un ensemble de propositions atomiques vraies dans cet état et $Q_i \subseteq Q$ est un ensemble non vide d'états initiaux.

Nous pouvons remarquer que tout $\Sigma^\star$-automate $\mathcal{A} = (S, \rightarrow, \Sigma, S_i, S_f)$ peut être traduit en une structure de Kripke $\mathcal{K}_\mathcal{A} = \langle Q, R, \lambda, Q_i \rangle$, avec :

- Q un ensemble d'états isomorphes à la relation de transition $\rightarrow$;
- $q_1 \, R \, q_2$ si il existe $s_1, s_2, s_3 \in S$ et $a, b \in \Sigma$ tels que $q_1 = s_1 \xrightarrow{a} s_2$ et $q_2 = s_2 \xrightarrow{b} s_3$;
- $\lambda(q) = \{a\}$ si $q = s \xrightarrow{a} s'$;
- $s \xrightarrow{a} s' \in Q_i$ si $s' \in S_i$.

Remarquons que $|\mathcal{K}_\mathcal{A}| = O(|\mathcal{A}|^2 \cdot |\Sigma|)$. Dans la suite de ce chapitre, nous remplacerons les structures de Kripke utilisées pour faire du model-checking par des $\Sigma^\star$-automates, pour obtenir des résultats moins généraux, mais qui ont plus de sens pour notre étude.

Linear Time Logic (LTL)

LTL ([Emerson 90]) est une logique temporelle qui permet d'exprimer des propriétés sur les exécutions d'un modèle. LTL utilise deux opérateurs modaux : X (next) et U (until). Etant donné un ensemble Σ de propositions atomiques, l'ensemble des formules de LTL_Σ, (ou simplement LTL), est défini par la grammaire suivante :

$$\varphi, \psi ::= \varphi U \psi \mid X\varphi \mid \varphi \wedge \psi \mid \neg\varphi \mid a \text{ (pour } a \in \Sigma)$$

Les autres opérateurs booléens : $\bot$, $\top$, $\varphi \vee \psi$, $\varphi \Rightarrow \psi$ et $\varphi \Leftrightarrow \psi$ sont définis de manière habituelle. La formule (X *pleuvoir*) peut être comprise intuitivement comme "Il va pleuvoir (juste après)" et (*heureux U pleuvoir*) comme "Je serais heureux jusqu'à ce qu'il pleuve". Les modalités F (correspondant à "peut-être dans le futur") et G ("toujours dans le futur") peuvent être obtenues par : $F\varphi = \top U \varphi$ et $G\varphi = \neg F \neg \varphi$.

Nous donnons maintenant la sémantique d'une formule LTL. Une telle formule est interprétée sur un chemin dans un automate. Etant donné un automate $\mathcal{A}$ et un chemin $\rho = n_0 \xrightarrow{\rho_1} \ldots \xrightarrow{\rho_{|\rho|}} n_{|\rho|}$ de $\mathcal{A}$, notons ρ_i la lettre étiquetant la i-ème transition de ρ et $|\rho|$ le nombre de transitions de ρ. La satisfiabilité d'une formule LTL_Σ φ par $\mathcal{A}$, ρ et un entier i, notée $\mathcal{A}, \rho, i \vDash \varphi$, est définie par induction sur φ :

- $\mathcal{A}, \rho, i \vDash a$ pour tout $a \in \Sigma$ si, et seulement si, $\rho_i = a$;
- $\mathcal{A}, \rho, i \vDash \phi \wedge \psi$ si, et seulement si, $\mathcal{A}, \rho, i \vDash \phi$ et $\mathcal{A}, \rho, i \vDash \psi$;
- $\mathcal{A}, \rho, i \vDash \neg\phi$ si, et seulement si, $\mathcal{A}, \rho, i \nvDash \phi$;
- $\mathcal{A}, \rho, i \vDash X\phi$ si, et seulement si, $i \neq |\rho|$ et $\mathcal{A}, \rho, i+1 \vDash \phi$;
- $\mathcal{A}, \rho, i \vDash \phi \; U \psi$ si, et seulement si, il existe un $j \geq i$ tel que $\mathcal{A}, \rho, j \vDash \psi$ et pour tout $i \leq k < j$, $\mathcal{A}, \rho, k \vDash \phi$.

De plus, nous noterons $\mathcal{A}, \rho \vDash \varphi$ à la place de $\mathcal{A}, \rho, 1 \vDash \varphi$, et $\mathcal{A} \vDash \varphi$ lorsqu'il existe un chemin ρ tel que $\mathcal{A}, \rho \vDash \varphi$. Etant donné un modèle $\mathcal{A}$ et une formule φ, le problème du model-checking est de vérifier si $\mathcal{A} \vDash \varphi$. Voici un exemple de formule LTL : $G(\neg \textit{requete} \vee (\textit{requete} \; U \; \textit{acceptation}))$ qui dit que, lorsqu'une requête est faite, elle reste active tant qu'elle n'est pas acceptée. De plus, le model-checking des formules LTL a une complexité raisonnable :

Théorème 3.1 ([Lichtenstein 85a, Sistla 85, Vardi 86a]) *Vérifier si un $\Sigma^\star$-automate $\mathcal{A}$ satisfait une formule LTL_Σ φ peut être déterminé avec une complexité temporelle en $O(|\mathcal{A}|^2 \cdot |\Sigma| \cdot 2^{O(|\varphi|)})$ ou une complexité spatiale en $O((|\varphi| + log(|\mathcal{A}| \cdot |\Sigma|))^2)$, avec $|\varphi|$ le nombre de caractères de la formule φ.*

De plus, en pratique, la taille des formules est petite comparée à la taille des modèles.

Computation Tree Logic* (CTL*)

CTL* étend LTL avec des quantificateurs sur les exécutions. En particulier, CTL* est plus expressive que LTL. Contrairement aux formules de LTL interprétées sur une unique exécution, les formules de CTL* sont interprétées sur l'ensemble des exécutions possibles du programme. Le quantificateur existentiel E ("il existe une exécution") sur les exécutions est ainsi ajouté aux opérateurs modaux de LTL.

La syntaxe d'une formule CTL* est donnée par la grammaire suivante :

$$\varphi, \psi ::= E\varphi \mid \varphi U \psi \mid X\varphi \mid \varphi \wedge \psi \mid \neg\varphi \mid a \text{ (pour } a \in \Sigma)$$

De plus, nous pouvons définir le quantificateur universel sur les chemins A ("pour tous les chemins") comme $A\varphi = \neg E \neg \varphi$.

Nous donnons maintenant la sémantique d'une formule LTL. Comme énoncé précédemment, une telle formule est interprétée sur l'ensemble des chemins d'un automate $\mathcal{A}$. Plus précisément, étant donné un automate $\mathcal{A}$, un chemin ρ et un entier i, la satisfiabilité d'une formule CTL* φ par $\mathcal{A}$, ρ et i, notée par $\mathcal{A}, \rho, i \vDash \varphi$, est définie par induction sur φ :

- $\mathcal{A}, \rho, i \vDash E\psi$ si, et seulement si, il existe un chemin ρ' de $\mathcal{A}$ tel que $\forall j\ 1 \leq j \leq i$, $\rho'_j = \rho_j$ et $\mathcal{A}, \rho', i \vDash \psi$;
- on utilise les même règles que pour LTL pour les autres constructeurs.

Et, de même que pour LTL, nous noterons $\mathcal{A} \vDash \varphi$ à la place de $\mathcal{A}, 1 \vDash \varphi$. Enfin, il existe des algorithmes, fondés sur les automates, qui résolvent le problème du model-checking pour les formules CTL* avec une complexité raisonnable :

Théorème 3.2 ([Emerson 87, Kupferman 00]) *Vérifier si un $\Sigma^\star$-automate $\mathcal{A}$ satisfait une formule CTL^*_Σ φ peut être déterminé avec une complexité temporelle en $O(|\mathcal{A}|^2 \cdot |\Sigma| \cdot 2^{O(|\varphi|)})$, avec $|\varphi|$ le nombre de caractères de la formule φ.*

La complexité de CTL* est donc la même que celle de LTL. Cependant, peu d'outils prenant en entrée des formules CTL* existent.

Computation Tree Logic (CTL)

CTL est un fragment de CTL* où chaque modalité temporelle (U ou X) doit être immédiatement précédée d'un quantificateur d'exécution (E ou A). La sémantique est la même que celle de CTL*. L'aspect intéressant de cette logique est la faible complexité de son model-checking.

Voici un exemple de formule CTL : $AG(requete \Rightarrow EF\ acceptation)$. Cette formule dit que, dès qu'un requête est faite, alors il existe une exécution où cette requête peut être acceptée.

Théorème 3.3 ([Clarke 86, Kupferman 00]) *Vérifier si un $\Sigma^\star$-automate $\mathcal{A}$ satisfait une formule CTL_Σ peut être déterminé avec une complexité temporelle en $O(|\mathcal{A}|^2 \cdot |\Sigma| \cdot |\varphi|)$ ou avec une complexité spatiale en $O(|\varphi| \cdot log(log(|\mathcal{A}| \cdot |\Sigma|))$, avec $|\varphi|$ le nombre de caractères de la formule φ.*

La figure 3.2, issue de [Schnoebelen 02], résume la complexité des trois principales logiques temporelles, en faisant la distinction entre la complexité liée à la taille du modèle à tester, la complexité liée à la taille de la formule à tester, et la complexité globale. Plus précisément, nous dirons que la complexité du model-checking liée à la taille du modèle est $\mathcal{C}$-complète, où $\mathcal{C}$ est une classe de complexité, si, d'une part, il existe une formule φ telle que, décider si $\mathcal{A} \vDash \varphi$ soit $\mathcal{C}$-difficile et, si, d'autre part, pour n'importe quelle formule φ, décider si $\mathcal{A} \vDash \varphi$ soit dans $\mathcal{C}$. Nous définissons, de la même manière, la complexité du model-checking liée à la taille de la formule.

	Modèle	Formule	Total
LTL	NLOGSPACE-complet	PSPACE-complet	PSPACE-complet
CTL	NLOGSPACE-complet	LOGSPACE	P-complet
CTL*	NLOGSPACE-complet	PSPACE-complet	PSPACE-complet

FIG. *3.2 – Complexité du model-checking pour LTL, CTL, et CTL*.*

Logiques d'ordres partiels

Nous donnons, maintenant, un bref aperçu des résultats existants lorsque la spécification du programme est décrite à l'aide d'un modèle de systèmes parallèles. Nous présentons tout d'abord des logiques de traces, qui ont vocation à spécifier des systèmes asynchrones. Ensuite, nous présentons différentes techniques de vérification de HMSC.

Logiques de traces

Une première approche concerne les logiques de traces qui étend le formalisme des logiques classiques, comme LTL, aux traces. Il existe deux approches pour ces logiques : celles qui décrivent le comportement des automates asynchrones comme LrPTL [Thiagarajan 94, Thiagarajan 02] et celle qui décrivent les chaînes d'événements dépendants. Cette dernière approche est elle-même divisée en deux parties, en fonction de la sémantique choisie pour l'opérateur décrivant ce qui va se passer après un événement. Dans le cas où l'opérateur choisi est existentiel (c'est à dire si l'on ne peut exprimer que des propriétés du type "il existe une chaîne de dépendance qui va me mener à un événement étiqueté par a") Alur et al. [Alur 95] ont défini la logique TLC, proche de LTL, dont la satisfiabilité et le model-checking sont dans PSPACE. Dans le cas où l'opérateur choisi est universel, Diekert et Gastin ont défini, dans [Diekert 01] la logique LocTL(EX,U), dont le problème de la satisfiabilité est PSPACE-complet. Cependant, cette logique n'utilise que deux modalités EX et U et ne permet pas d'exprimer toutes les formules du premier ordre. Pour pallier ce problème, Gastin et Mukund ont proposé, dans [Gastin 02], une logique qui modifie la modalité U de LocTL(EX,U) pour être plus expressive et conserver la complexité de la satisfiabilité.

Ces approches mettent en jeu des techniques de preuve très différentes. Cependant, Gastin et Kuske ont montré dans [Gastin 03] que tous ces problèmes pouvaient se résoudre à la satisfiabilité et au model-checking de logiques temporelles définissables en MSO, ce qu'ils ont montré être des problèmes qui sont dans PSPACE.

HMSC et logiques temporelles classiques

Tout d'abord, dans le cadre des HMSC, le problème du model-checking devient difficile. En effet, les techniques développées sur les automates utilisent le fait suivant : tester si $\mathcal{A} \vDash \varphi$ est équivalent à tester $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}_\varphi)$, pour un $\Sigma^\star_{\mathcal{P},\mathcal{M},\mathcal{I}}$-automate $\mathcal{A}_\phi$ qui génère tous les modèles satisfaisant ϕ. Ceci est équivalent à tester $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}_{\bar{\varphi}}) = \emptyset$, avec $\mathcal{A}_{\bar{\varphi}}$ l'automate dont le langage est le complément du langage de $\mathcal{A}_\varphi$ dans $\Sigma^\star$. Dans le cadre des HMSC, la situation est plus complexe. En effet, le complément du langage d'un HMSC ne peut pas être représenté à l'aide d'un HMSC (car le complément n'est pas finiment engendré), et donc, dans le cas général, on ne peut pas utiliser la construction précédente. De plus, Muscholl, Peled et Su ont montré dans [Muscholl 98] que l'inclusion et l'intersection de HMSC étaient indécidables. Tout ceci implique que le problème du model-checking est indécidable pour les HMSC. De même, Alur et Yannakakis dans [Alur 99] ont montré que LTL était indécidable pour les HMSC.

Cependant, lorsque l'on limite le pouvoir d'expression de la logique employée ou le pouvoir d'expression des modèles, nous obtenons à nouveau des résultats de décidabilité pour le model-checking.

D'une part, Peled dans [Peled 00] a limité le pouvoir d'expression de la logique en proposant une logique proche de LTL pour les HMSC, nommée TLC^- (c'est une restriction de la logique TLC définie pour les traces). Cette logique est locale et, plutôt que de décrire le langage de linéarisations des MSC générés par un HMSC, elle permet de décrire des chaînes de causalité dans ces MSC. Elle permet par exemple de spécifier le comportement d'un unique processus ou de définir une chaîne causale de messages. Plus précisément, elle permet de spécifier des chaînes d'événements reliés par $\prec$, la relation de couverture de la relation de causalité définie dans les MSC considérés. Dans ce cadre, le problème du model-checking de HMSC est décidable. Ces résultats de décidabilité peuvent s'étendre aux CHMSC.

De même, lorsque la formule à tester est représentée sous la forme d'un HMSC globalement coopératif, deux problèmes de model-checking deviennent décidable. Plus précisément, le premier problème est le model-checking positif, qui consiste à vérifier qu'un modèle, α, ne génère que des comportements admissibles, se ramène à un problème d'inclusion $\mathcal{L}(\alpha) \subseteq \mathcal{L}(\beta)$, avec β un HMSC qui génère les comportements admissibles. Le model-checking positif de HMSC et de CHMSC globalement coopératifs est EXPSPACE-complet[Genest 02b, Genest 06a]. Le second problème est le model-checking négatif, qui consiste à vérifier qu'un modèle, α, ne génère aucun comportement non désiré, se ramène à un problème de vacuité d'intersection $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta) = \emptyset$, avec β un HMSC qui génère les comportements non désirés. Le model-checking négatif de HMSC et de CHMSC globalement coopératifs est PSPACE-complet[Genest 02b, Genest 06a]. La complexité de ces deux problèmes de model-checking découle directement de la proposition 1.33.

D'autre part, lorsque les modèles sont des HMSC réguliers le model-checking redevient décidable, pour n'importe quelle logique définie pour les systèmes séquentiels. En effet, les HMSC réguliers ont des langages de linéarisations réguliers et peuvent donc être représentés à l'aide d'un automate. Les logiques permettent alors d'exprimer des propriétés sur les linéarisations d'un HMSC donné. On peut noter aussi que le model-checking négatif de HMSC globalement coopératif reste aussi décidable.

Template HMSC

Ensuite, Genest et al. ont introduit dans [Genest 04a] une logique fondée sur des motifs décrits à l'aide de MSC qui possèdent des "trous", ces trous servant de filtres à certaines actions. Ce nouveau modèle, nommé "template MSC", consiste donc en des MSC incomplets. De plus, une logique est définie sur ce modèle, à l'aide de pré et post-conditions, et de formules de conjonction et de disjonction. Plus précisément, les pré et post-conditions sont des opérateurs de type $M_a \longrightarrow M_b$ et $M_a \longrightarrow \neg M_b$, où M_a et M_b sont des template MSC. Les formules de cette logique sont de la forme $\wedge_i(M_a^i \longrightarrow (\vee_j \widehat{M}^{ij}))$ où $\widehat{M}^{ij}$ est, soit un littéral positif M_b^{ij}, soit un littéral négatif $\neg M_b^{ij}$.

La figure 3.3 donne un exemple de formule de template MSC qui spécifie le protocole WRS (Writer-Reader-Server). Cette formule indique que lorsque le serveur envoie une valeur à un lecteur, cet envoi est suivi d'une validation.

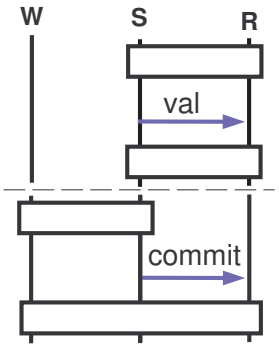


FIG. *3.3 – Exemple de formule de template* MSC *:* $M_a \longrightarrow \neg M_b$.

Cette logique étant expressive, la satisfiabilité d'une formule n'est pas décidable. Cependant, lorsqu'on se limite aux modèles existentiellement bornés (ce qui est le cas quand on se limite aux HMSC) elle devient décidable, de même que le model-checking. Ces travaux sont à rapprocher de ceux sur les Live MSC([Damm 99]) ou les Triggered MSC([Sengupta 02]) qui visent à modifier les HMSC pour leur permettre d'exprimer des propriétés plus pertinentes.

3.2 Model-checking d'expressions rationnelles de pomsets

Dans la suite de ce chapitre, nous modifions quelque peu le problème du model-checking défini dans la partie précédente, afin de l'adapter aux modèles que nous manipulons. Nous

considérons donc que la spécification d'un programme est donnée sous la forme d'une expression corationnelle de REX($\mathbb{P}(\Sigma, D)$), et que la formule à vérifier est construite à partir de propositions atomiques qui sont des pomsets, exprimant des comportements possibles (ou impossibles) de la spécification. Ainsi, pour toute spécification α et formule φ, nous notons $\alpha \vDash \varphi$ à la place de $\mathcal{A}_\alpha \vDash \varphi$, où $\mathcal{A}_\alpha$ est l'automate structurellement semblable à α.

Nous commençons par définir la sémantique d'une formule d'ordres partiels, exprimée dans $\mathrm{CTL}^*_{\mathbb{P}(\Sigma,D)}$. Pour cela, considérons α, un modèle exprimé à l'aide d'une expression rationnelle de pomsets, $\rho = n_0 \xrightarrow{\rho_1} \dots \xrightarrow{\rho_{|\rho|}} n_{|\rho|}$, un chemin de $\mathcal{A}_\alpha$ et i, un entier. La satisfiabilité d'une formule φ de $\mathrm{CTL}^*_{\mathbb{P}(\Sigma,D)}$ par $\mathcal{A}$, ρ et i est définie par induction sur φ :

- $\alpha, \rho, i \vDash p$ si, et seulement si, $\rho_i = p$ pour tout $p \in \mathbb{P}(\Sigma, D)$;
- $\alpha, \rho, i \vDash \phi \wedge \psi$ si, et seulement si, $\alpha, \rho, i \vDash \phi$ et $\alpha, \rho, i \vDash \psi$;
- $\alpha, \rho, i \vDash \neg\phi$ si, et seulement si, $\alpha, \rho, i \nvDash \phi$;
- $\alpha, \rho, i \vDash X\phi$ si, et seulement si, $i \neq |\rho|$ et $\alpha, \rho, i+1 \vDash \phi$;
- $\alpha, \rho, i \vDash \phi\ U\psi$ si, et seulement si, il existe un $j \geq i$ tel que $\alpha, \rho, j \vDash \psi$ et pour tout $i \leq k < j$, $\alpha, \rho, k \vDash \phi$;
- $\alpha, \rho, i \vDash E\psi$ si, et seulement si, il existe un chemin ρ' de $\mathcal{A}_\alpha$ tel que $\forall j\ 1 \leq j \leq i$, $\rho'_j = \rho_j$ et $\alpha, \rho', i \vDash \psi$.

Nous noterons $\alpha \vDash \varphi$ si il existe un chemin ρ de $\mathcal{A}_\alpha$ tel que $\mathcal{A}, \rho, 1 \vDash \varphi$. De la même manière, il est possible de définir $\mathrm{LTL}_{\mathbb{P}(\Sigma,D)}$ et $\mathrm{CTL}_{\mathbb{P}(\Sigma,D)}$. De manière intuitive, cette logique permet de spécifier des propriétés sur le langage de linéarisations de pomsets qui ne sont pas décomposables (et qui seront appelés *premiers* dans la partie 3.2.1). Quand le modèle engendre un langage de pomsets premiers qui est clos par commutation, cette logique prend tout son sens puisqu'elle permet d'exprimer des propriétés temporelles fondées sur la causalité entre pomsets premiers : en effet, deux pomsets premiers peuvent commuter si, et seulement si, leur composition séquentielle n'ajoute aucune causalité entre ces deux mêmes pomsets. Cependant, cette logique ne permet pas d'exprimer directement des causalités entre les événements.

Les problèmes auxquels nous nous intéressons dans la suite de ce chapitre sont les suivants :

Model-checking de formules d'ordres partiels dans REX($\mathbb{P}(\Sigma, D)$)
Entrées :
- un modèle α, exprimé à l'aide d'une expression rationnelle de $\mathbb{P}(\Sigma, D)$;
- une formule φ à tester, exprimée à l'aide d'une formule de $\mathrm{LTL}_{\mathbb{P}(\Sigma,D)}$, $\mathrm{CTL}_{\mathbb{P}(\Sigma,D)}$, ou $\mathrm{CTL}^\star_{\mathbb{P}(\Sigma,D)}$.

Sortie : Est-ce que $\alpha \vDash \varphi$?

Model-checking positif dans REX($\mathbb{P}(\Sigma, D)$)
Entrées :
- un modèle α, exprimé à l'aide d'une expression rationnelle de $\mathbb{P}(\Sigma, D)$;
- un modèle β de bons comportements, exprimé à l'aide d'une expression rationnelle de $\mathbb{P}(\Sigma, D)$ (β est abusivement appelée formule positive).

SORTIE : Est-ce que $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha) \subseteq \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\beta)$?

Model-checking négatif dans $\mathrm{REX}(\mathbb{P}(\Sigma, D))$
ENTRÉES :
- un modèle α, exprimé à l'aide d'une expression rationnelle de $\mathbb{P}(\Sigma, D)$;
- un modèle β de mauvais comportements, exprimé à l'aide d'une expression rationnelle de $\mathbb{P}(\Sigma, D)$ (β est abusivement appelée formule négative).

SORTIE : Est-ce que $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha) \cap \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\beta) = \emptyset$?

Le premier résultat que nous pouvons donner est un résultat négatif. En effet, rappelons encore une fois que la vacuité de l'intersection des langages de deux HMSC est indécidable (prop. 1.33), ce qui entraîne directement l'indécidabilité des problèmes de model-checking dans $\mathrm{REX}(\mathbb{P}(\Sigma, D))$, auxquels nous nous intéressons. Dans la suite de ce chapitre, nous allons donc nous intéresser à des restrictions syntaxiques (et donc décidables) des spécifications et des formules logiques pour lesquelles nous allons effectivement pouvoir résoudre ces problèmes.

La figure 3.4 donne un exemple de formule dans $\mathrm{LTL}_{\mathbb{P}(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}},D)}$ (que l'on peut interpréter par des HMSC causaux), avec une indépendance de tous les événements sur le processus p :

$$I_p = \{(p!q(req), p!r(req)), (p!r(req), p!q(req)), (p?q(ack), p?r(ack)), (p?r(ack), p?q(ack))\}$$

La formule qui y est représentée indique que, dès qu'un serveur p envoie une requête à deux clients q et r, il continue à envoyer ces requêtes jusqu'à ce qu'un des deux clients réponde. Remarquons que le langage de linéarisations de cette formule n'est pas régulier et que nous utilisons une négation.

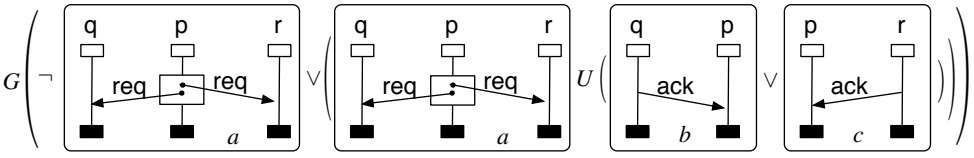


FIG. *3.4 – Exemple de formule dans* $LTL_{\mathbb{P}(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}},D)}$.

Cette partie est organisée de la manière suivante. Tout d'abord, nous donnons, dans la partie 3.2.1 une méthode pour décomposer canoniquement un pomset en un produit de pomsets "indivisibles". Ensuite, cette méthode nous permettra de plonger le monoïde des pomsets dans le monoïde des traces, dans la partie 3.2.2. Enfin, grâce à cette traduction, nous verrons, dans la partie 3.2.3 que les problèmes de model-checking définis ci-dessus sont décidables pour des sous-classes syntaxiques du monoïde des pomsets.

3.2.1 Pomsets premiers

Nous allons maintenant nous inspirer des résultats connus sur les HMSC globalement coopératifs pour trouver une sous-classe syntaxique de $\mathrm{REX}(\mathbb{P}(\Sigma, D))$, l'ensemble des expressions rationnelles dont les éléments générateurs sont des pomsets étiquetés par Σ, sur laquelle le model-checking est décidable. Pour cela, nous allons adapter la notion d'atomes

de [Ahuja 90, Hélouët 00b].

Tout d'abord, nous généralisons la notion de MSC atomique à celle de pomset *premier*.

Définition 3.4 *Un pomset $p \in \Gamma$ est* premier *(vis à vis de la relation de dépendance D) s'il ne peut pas être décomposé en deux pomsets non-vides $p_1, p_2 \in \mathbb{P}(\Sigma, D) - \{\varepsilon\}$ tels que $p = p_1 \circledcirc_D p_2$.*

Nous montrons ensuite que la décomposition en éléments premiers d'un pomset $p = [E, \leq, \lambda]$ peut se réduire à construire le quotient de $\leq$ par une relation d'équivalence appropriée. Cette approche est constructive, puisqu'elle peut se réduire à la recherche de composants fortement connexes dans un graphe construit à partir de E, ce qui est résoluble en temps quadratique d'après [Tarjan 72]. L'algorithme que nous donnons, s'inspire de celui proposé dans [Hélouët 00b], qui calcule la décomposition en atomes d'un MSC quelconque. L'intuition derrière ce résultat est la suivante : les éléments fortement connectés par R sont des événements qui doivent appartenir au même pomset premier, car il n'est pas possible de les partitionner plus finement sans perdre de causalité lors de leur recomposition par $\circledcirc_D$.

Proposition 3.5 *Soient $p = [E, \leq, \lambda]$, un pomset, $\preceq$, la relation de couverture de $\leq$ et R, la relation qui vérifie $e\, R\, e'$ si :*

- *$e \preceq e'$;*
- *ou $e' \preceq e$ et $\lambda(e)\, I\, \lambda(e')$;*
- *ou $e \not\leq e'$, $e' \not\leq e$ et $\lambda(e)\, D\, \lambda(e')$.*

Alors, les deux problèmes suivants sont équivalents.

1. *décomposer le pomset p en $p_1 \circledcirc_D \ldots \circledcirc_D p_k$, où $p_i = [E_i, \leq_i, \lambda_i]$ est un pomset premier pour tout $1 \leq i \leq k$;*
2. *décomposer le graphe $\mathcal{G}_p = (E, R)$ en composantes fortement connexes $\{E_i\}_{1 \leq i \leq k}$ avec $\forall i < j$, $(E_j \times E_i) \cap R = \emptyset$.*

Preuve: Posons $p = [E, \leq, \lambda]$ un pomset non vide, et notons $R^{-1} = \{(e, e') \mid (e', e) \in R\}$, $R_I = \{(e', e) \in E^2 \mid e' \preceq e \wedge \lambda(e)\, I\, \lambda(e')\}$ et $R_D = \{(e, e') \in E^2 \mid e \not\leq e' \wedge e' \not\leq e \wedge \lambda(e)\, D\, \lambda(e')\}$. Nous pouvons donc maintenant réécrire R de la manière suivante : $R = \preceq \cup R_I \cup R_D$.

D'une part, montrons que $\neg(1.) \Rightarrow \neg(2.)$. Prenons deux pomsets non vides $p_1 = [E_1, \leq_1, \lambda_1]$ et $p_2 = [E_2, \leq_2, \lambda_2]$ tels que $p = p_1 \circledcirc_D p_2$ (de tels p_1 et p_2 existent si p n'est pas premier). Nous pouvons alors remarquer que $R_I \cap (E_2 \times E_1 = \emptyset$ et $R_D \cap (E_2 \times E_1) = \emptyset$. Par conséquent, $R \cap (E_2 \times E_1) = \emptyset$ et par conséquent E n'est pas une composante fortement connexe de $\mathcal{G}_p$.

D'autre part, montrons que $\neg(2.) \Rightarrow \neg(1.)$. Supposons que E ne soit pas une composante fortement connexe de $\mathcal{G}_p$. Cela signifie que nous pouvons décomposer E en $n > 1$ composantes fortement connexes $E_1, \ldots, E_n$. Nous étendons, ensuite, la définition de R aux couples de sous-ensembles (E_i, E_j) tels qu'il existe $(e_i, e_j) \in E_i \times E_j$ avec $(e_i, e_j) \in R$. Cette définition est équivalente à : $(E_i, E_j) \in R$ si, et seulement si, il existe $(e_i, e_j) \in E_i \times E_j$ avec $(e_i, e_j) \in R - R^{-1}$. En effet, si (e_i, e_j) est dans R^{-1}, alors E_i et E_j forment une unique composante, ce qui n'est pas le cas. De plus, comme $R_I \cap (R_D \cup R_D^{-1}) = \emptyset$, $\preceq \cap (R_D \cup R_D^{-1}) = \emptyset$ et $R_I \subseteq \preceq^{-1}$, nous

avons $R - R^{-1} = \preceq - R_I^{-1} = \{(e, e') \mid e \preceq e' \wedge \lambda(e) D \lambda(e')\}$. Ainsi, $(E_i, E_j) \in R$ si, et seulement si, pour tout $(e_i, e_j) \in E_i \times E_j$, $e_i \preceq e_j$ implique que $\lambda(e_i) D \lambda(e_j)$. Finalement, nous pouvons déduire de l'affirmation précédente que $R \cap \preceq \cap (E_i \times E_j) = \{(e_i, e_j) \in E_i \times E_j \mid \lambda(e_i) D \lambda(e_j)\}$, ce qui signifie que $p_{E_i} \circledcirc_D p_{E_j} = p_{E_i \cup E_j}$, avec $p_{E'}$ le pomset résultant de la projection de p sur les événements $E' \subseteq E$. Enfin, pour terminer la preuve, il faut considérer une linéarisation $E_{i_1} \dots E_{i_n}$ des n composantes fortement connexes de E, puis la composition $p_{E_{i_1}} \circledcirc_D p_{E_{i_2} \cup \dots E_{i_n}}$ qui est donc égale à p, ce qui signifie que p n'est pas un pomset premier. □

Cette proposition donne donc un algorithme direct de décomposition en pomsets premiers : étant donné un pomset $p = [E, \leq, \lambda]$, il faut :

1. construire le graphe $\mathcal{G}_p = (E, R)$ de la proposition 3.5 ;
2. calculer les composantes fortement connexes $E_1 \dots E_n$ de $\mathcal{G}_p$;
3. choisir une linéarisation $l = p_{i_1} \dots p_{i_n}$ telle que p_j soit la projection de p sur E_j et $k < l$ implique que pour tout $(e_k, e_l) \in E_k \times E_l$, $(e_k, e_l) \notin R$.

Ainsi, en utilisant ces résultats, il est possible de transformer, en temps quadratique, toute expression de $\mathrm{REX}(\mathbb{P}(\Sigma, D))$ en expression rationnelle sur des pomsets premiers. Dans la suite de ce chapitre, nous supposons donc que nous pouvons traduire n'importe quelle expression rationnelle de pomsets en une expression rationnelle de pomsets premiers ayant un langage équivalent.

3.2.2 Langages corationnels de pomsets

De la même manière que [Morin 02] l'a fait pour les HMSC, nous allons plonger $\mathrm{REX}(\mathbb{P}(\Sigma, D))$ dans le monoïde de traces. Pour cela, nous associons naturellement à $\mathbb{P}(\Sigma, D)$, une relation d'indépendance $\|$ définie de la manière suivante : pour tout pomset p et q, $p \parallel q$ si pour tout $a \in \Sigma(p)$ et $b \in \Sigma(q)$, $a\ I\ b$, c'est-à-dire si les alphabets étiquetant p et q sont totalement indépendants. Clairement, $p \parallel q$ implique que $p \circledcirc_D q = q \circledcirc_D p$. Etant donné un ensemble fini Γ de pomsets, notons $\not\parallel = \Gamma^2 - \parallel$.

Théorème 3.6 (isomorphisme avec le monoïde de traces) *Soit Γ, un ensemble fini de pomsets premiers. Alors, le morphisme $\eta : \mathbb{M}(\Gamma, \not\parallel) \to \Gamma^\star$, qui associe à chaque trace $[a_1 \dots a_n]_{\sim_\parallel}$ le pomset $a_1 \circledcirc_D \dots \circledcirc_D a_n$ est un isomorphisme.*

Preuve: Tout d'abord, η est bien un morphisme de monoïdes puisque :

$$\eta([pq]_{\sim_\parallel}) = \eta([p]_{\sim_\parallel} \cdot [q]_{\sim_\parallel}) = p \circledcirc_D q = \eta([p]_{\sim_\parallel}) \circledcirc_D \eta([q]_{\sim_\parallel})$$

Ensuite, montrons que si $\eta(u) = \eta(v)$, alors $u \sim_\parallel v$. Pour cela, montrons que la décomposition en atomes de v, interprétée dans $\mathbb{P}(\Sigma, D)$, est unique, à commutation près. C'est à dire, toute décomposition $a_1 \dots a_k$ d'un pomset p appartient à $[a_1 \dots a_k]_{\sim_\parallel}$. Il suffira ensuite de décomposer u et v de manière canonique pour prouver le résultat voulu.

Supposons le contraire, c'est-à-dire qu'il existe un mot $w = b_1 \dots b_q$, composé de pomsets premiers, tels que $p = b_1 \circledcirc_D \dots \circledcirc_D b_q$ mais que $w \notin [a_1 \dots a_k]_{\sim_\parallel}$. La proposition 3.5 montre que $\{a_1, \dots, a_k\}$ est la plus petite partition induite par la relation d'équivalence $\sim$, définie comme suit : $e \sim e'$ si, et seulement si, e et e' appartiennent à la même composante fortement connexe du graphe $\mathcal{G}_p$. Ce qui implique que $\{b_1, \dots, b_q\} = \{a_1, \dots, a_k\}$. On peut

donc trouver une fonction $f : \{b_1, \ldots, b_q\} \to \{a_1, \ldots, a_k\}$ qui associe à chaque b_i un premier de $\{a_1, \ldots, a_k\}$ qui lui est isomorphe. De plus, si $w \notin [a_1 \ldots a_k]_{\sim_\parallel}$, alors il existe deux pomsets premiers de w, appelons les b_i et b_j, tels que b_i précède b_j dans w, et pour toute fonction f comme définie précédemment, $f(b_j)$ précède $f(b_i)$ dans $a_1 \ldots a_k$, et $f(b_i) \not\parallel f(b_j)$. On peut donc trouver deux événements e_i et e_j respectivement dans $f(b_i)$ et $f(b_j)$ tels que $\lambda(e_i)\, D\, \lambda(e_j)$. Or $\Sigma(b_i) = \Sigma(f(b_i))$, donc $b_i \not\parallel b_j$, et donc b_i ne peut précéder b_j. Contradiction. □

Nous pouvons maintenant tirer les conséquences du théorème 3.6, et appliquer les résultats connus de décidabilité sur les traces corationnelles. Pour cela, nous identifions une sous-classe de $\mathrm{REX}(\mathbb{P}(\Sigma, D))$, appelée corationnels de pomsets, qui correspond exactement aux langages de traces corationnels.

Théorème 3.7 (reconnaissabilité) *Soient (Σ, D), un alphabet concurrent, α, une expression rationnelle de REX$(\mathbb{P}(\Sigma, D))$ et η, l'isomorphisme défini dans le théorème 3.6.*

α est une expression corationnelle *de REX$(\mathbb{P}(\Sigma, D))$ si, pour toute sous-expression $\beta^\star$ de α, $\eta^{-1}(\beta)$ est une trace $\not\parallel$-connexe. De manière équivalente, α est une expression* corationnelle *de REX$(\mathbb{P}(\Sigma, D))$ si, pour toute sous-expression $\beta^\star$ de α, $(\Gamma(\beta), \not\parallel)$ est un graphe connexe, où $\Gamma(\alpha)$ est l'ensemble des pomsets premiers qui sont utilisés dans l'expression rationnelle α.*

Si α est corationnelle, alors $\mathcal{L}(\alpha)$ est reconnaissable dans $\mathbb{P}(\Sigma, D)$. Plus précisément, pour toute expression corationnelle α, il existe un automate $\mathcal{A}$ tel que $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\mathcal{A}) = \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha)$ et $\mathcal{L}_{\Gamma(\alpha)^\star}(\mathcal{A}) = [\mathcal{L}_{\Gamma(\alpha)^\star}(\alpha)]_{\sim_\parallel}$ avec $|\mathcal{A}| = 2^{O(|\alpha| \cdot |\Gamma(\alpha)|)}$.

Preuve: Nous allons utiliser l'isomorphisme η du théorème 3.6, pour plonger $\mathrm{REX}(\mathbb{P}(\Sigma, D))$ dans le monoïde de traces $\mathbb{M}(\Gamma(\alpha), \not\parallel)$. Tout d'abord, en utilisant η, nous pouvons remarquer que l'ensemble des traces corationnelles de $\mathbb{M}(\Gamma(\alpha), \not\parallel)$ est isomorphe à l'ensemble des expressions corationnelles de $\mathrm{REX}(\mathbb{P}(\Sigma, D))$.

Ensuite, nous appliquons le résultat d'Ochmanski sur les traces corationnelles (th. 1.12) pour obtenir que les corationnels de $\mathbb{M}(\Gamma(\alpha), \not\parallel)$ sont reconnaissables. Puis, nous utilisons le fait que η soit un morphisme, pour en conclure que les expressions corationnelles de $\mathrm{REX}(\mathbb{P}(\Sigma, D))$ sont reconnaissables.

Pour conclure la démonstration de ce théorème, nous appliquons le résultat de complexité de [Muscholl 99] pour obtenir que la taille de l'automate $\mathcal{A}$ est exponentielle par rapport à la taille de α et au nombre de pomsets premiers utilisés pour générer α. □

Le théorème 3.7 va nous servir de base pour tous les résultats suivants. En effet, pour toute spécification corationnelle de pomsets, nous pouvons, maintenant, construire une structure opérationnelle ayant le même comportement. En utilisant cette structure, il est alors possible de transposer et d'appliquer directement les nombreux résultats exposés dans l'état de l'art (partie 3.1). C'est ce que nous allons voir dans la partie suivante.

3.2.3 Model-checking de corationnels de pomsets

Nous allons utiliser le résultat de la partie précédente pour répondre aux trois types de model-checking auxquels nous nous intéressons dans ce chapitre, à savoir le model-checking

de formules d'ordres partiels, le model-checking positif, et le model-checking négatif.

Comme nous l'avons vu précédemment, dans le cadre général, résoudre ces trois problèmes est indécidable. Nous allons donc donner des restrictions syntaxiques qui permettent de rester dans un cas décidable. La première restriction concerne les langages utilisés pour décrire les modèles, la seconde concerne les langages qui sont utilisés pour décrire la formule à tester.

Restriction de la puissance d'expression des modèles

D'une part, nous résolvons le problème du model-checking de formules d'ordres partiels lorsque la spécification du modèle est corationnelle et la formule logique est quelconque. Bien que limitant le pouvoir d'expression des spécifications, ce résultat nous semble cependant intéressant. En effet, il est possible d'utiliser des expressions corationnelles de $\mathrm{REX}(\mathbb{P}(\Sigma, D))$ afin de spécifier des systèmes dont le langage des entrelacements n'est pas régulier, et ensuite, de les tester vis-à-vis de n'importe quelle logique temporelle.

Proposition 3.8 (Model-checking de formules d'ordres partiels) *Soient α, une spécification corationnelle, $\mathcal{A}_\alpha$, l'automate structurellement équivalent et φ, une formule de logique temporelle, dont les propositions atomiques correspondent à des pomsets premiers. Alors, $\mathcal{A}_\alpha \vDash \varphi$ est décidable et, de plus :*
(1) si φ est une formule de $LTL_{\mathbb{P}(\Sigma,D)}$, alors décider si $\mathcal{A}_\alpha \vDash \varphi$ est PSPACE-complet ;
(2) si φ est une formule de $CTL_{\mathbb{P}(\Sigma,D)}$, alors décider si $\mathcal{A}_\alpha \vDash \varphi$ est dans PSPACE ;
(3) si φ est une formule de $CTL^*_{\mathbb{P}(\Sigma,D)}$, alors décider si $\mathcal{A}_\alpha \vDash \varphi$ est PSPACE-complet.

Preuve: Tout d'abord, les bornes inférieures sont obtenues dans les théorèmes 3.1, 3.2 et 3.3. Ensuite, α étant corationnelle, nous pouvons appliquer le théorème 3.7 pour obtenir un automate $\mathcal{A}$ de taille en $2^{O(|\alpha|\cdot|\Gamma(\alpha)|)}$ tel que $\mathcal{L}_{\Gamma(\alpha)^\star}(\mathcal{A}) = [\mathcal{L}_{\Gamma(\alpha)^\star}(\alpha)]_{\sim_\parallel}$. Nous pouvons donc remplacer les automates des théorèmes 3.1, 3.2 et 3.3 par ce nouvel automate $\mathcal{A}$, ce qui ne modifie pas la classe de complexité du problème : en effet, pour une formule fixée, la classe de complexité du model-checking (qui ne dépend donc que de la taille du programme) est NLOGSPACE-complet (voir la figure 3.2). □

Restriction du pouvoir d'expression des formules

D'autre part, nous proposons une solution pour résoudre les problèmes du model-checking positif et négatif, dans le cadre où les modèles positifs ou négatifs à tester sont des langages corationnels de pomsets.

Proposition 3.9 (Model-checking positif et négatif) *Soient α et β, deux expressions rationnelles de $REX(\mathbb{P}(\Sigma, D))$. Si β est corationnelle, alors :*
Model-checking positif : décider si $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha) \subseteq \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\beta)$ est EXPSPACE-complet ;
Model-checking négatif : décider si $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha) \cap \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\beta) = \emptyset$ est PSPACE-complet.

Preuve: Tout d'abord, le théorème 3.7 nous donne un automate $\mathcal{B}$ tel que $[\mathcal{L}_{\Gamma(\beta)^\star}(\beta)]_{\sim_\parallel} = \mathcal{L}_{\Gamma(\beta)^\star}(\mathcal{B})$, et la taille de $\mathcal{B}$ est exponentielle en celle de β. Ensuite, il suffit de remarquer que pour toute congruence $\sim$ et tout langage L et L', tester $[L]_\sim \subseteq [L']_\sim$ est équivalent à tester

$L \subseteq [L']_\sim$. De même, tester $[L]_\sim \cap [L']_\sim = \emptyset$ est équivalent à tester $L \cap [L']_\sim = \emptyset$. Finalement, les bornes minimales viennent de [Muscholl 99] qui montre ce résultat sur les HMSC, et les bornes maximales viennent de la complexité de ces opérations sur des automates de taille exponentielle par rapport à la taille des structures initiales. □

Nous avons donc proposé deux restrictions syntaxiques qui permettent de répondre aux trois problèmes de model-checking considéré. Tout d'abord, nous avons montré que le model-checking d'ordres partiels est décidable, avec les mêmes complexités que pour son équivalent séquentiel, lorsque le modèle est un langage corationnel de pomsets. Ce résultat est très satisfaisant, puisqu'il montre que l'on peut gagner en concision (grâce à la modélisation utilisant des ordres partiels) sans pour autant perdre de l'efficacité. Ensuite, nous avons montré qu'en limitant les formules négatives et positives aux langages corationnels de pomsets, nous pouvons résoudre le model-checking positif et négatif pour n'importe quel type de modèle. C'est aussi un résultat intéressant puisqu'il montre qu'il est possible d'analyser *toutes* les spécifications écrites dans ce langage.

Exemple

La figure 3.5 montre un exemple de modèle qui peut être testé par rapport à la propriété définie dans la figure 3.4. Ce modèle correspond à un HMSC causal dont la relation de dépendance est la même que celle définie pour la propriété, à savoir seul les deux envois des messages *req* sont indépendants.

Ce modèle représente deux clients, qui envoient chacun des données à un serveur. Puis celui-ci envoie une requête à chacun des clients. Enfin, un seul client envoie une réponse au serveur. Ce comportement est ensuite répété un nombre arbitraire de fois.

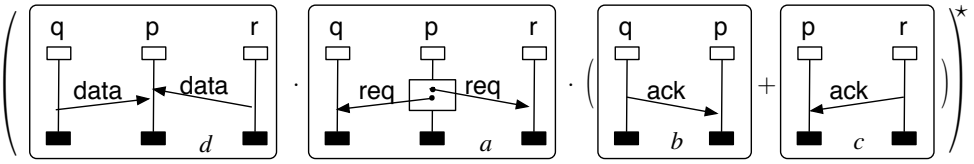


FIG. *3.5 – Exemple de d'expression corationnelle de pomset.*

Etant donné le modèle α, défini dans la figure 3.5, avec une relation utilisée pour la composition qui rend dépendants tous les événements qui ont lieu sur q et tous les événements ayant lieu sur r et indépendants tous les autres et la formule LTL φ, définie dans la figure 3.4, nous pouvons remarquer que $\alpha \vDash \varphi$. En effet, le langage des pomsets premiers de α peut être engendré par l'expression rationnelle $(da(b+c))^\star$, qui est corationnelle car tous ses cycles sont $\not\|$-connexes (les événements sur q et r étant dépendants entre eux) et dont la clôture par $||$ est engendrée par la même expression rationnelle. Il suffit alors de tester, en utilisant les méthodes classiques de model-checking, que :

$$(da(b+c))^\star \vDash G(\neg a \vee (aU(b \vee c)))$$

3.3 Model-checking de HMSC causaux

Nous allons maintenant appliquer les résultats montrés dans la partie précédente aux HMSC causaux. Comme montré sur les exemples, la traduction est immédiate. Nous nous contentons donc d'énumérer les résultats et de comparer les classes où le model-checking est décidable avec d'autres modèles qui étendent les HMSC.

3.3.1 Différentes sémantiques pour les HMSC causaux

Avant de préciser les résultats précédents pour le cas particulier des HMSC causaux, rappelons qu'un HMSC causal α est une structure qui permet de définir de manière syntaxique (au moins) trois langages différents, à savoir :

1. le langage des linéarisations : $Lin(\alpha) \subseteq \Sigma^{\star}_{\mathcal{P},\mathcal{M},\mathcal{I}}$;
2. la langage des MSC : $\text{MSC}(\alpha) \subseteq \mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})$;
3. le langage des MSC causaux $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\alpha) \subseteq \mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)$.

La proposition suivante montre que ces trois langages sont, en général, sémantiquement différents. Elle identifie aussi les restrictions sous lesquelles ces différents langages coïncident.

Proposition 3.10 *Soient $(\Sigma_{\mathcal{P},\mathcal{M},\mathcal{I}}, D)$, un alphabet concurrent et localisé, α et β, deux* HMSC *causaux. Considérons les hypothèses suivantes :*

1. *α et β ont le même langage de* MSC *causaux : $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\alpha) = \mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\beta)$*
2. *α et β ont le même langage de* MSC *: $\text{MSC}(\alpha) = \text{MSC}(\beta)$*
3. *α et β ont le même langage de linéarisations : $Lin(\alpha) = Lin(\beta)$*

Alors, nous avons les résultats suivants :
- *1⇒2 mais, en général, l'inverse n'est pas vrai ;*
- *3 ⇔ 2 ;*
- *Si pour tout $m \in \Gamma(\alpha) \cup \Gamma(\beta)$, m est D-cohérent (cf. définition 2.11), alors 2⇒1.*

Preuve: Tout d'abord, les implications 1⇒2 et2⇒3 sont des conséquences directes des définitions : en effet, le langage des linéarisations est obtenu à partir du langage des MSC qui est lui même obtenu à partir du langage des MSC causaux.

Ensuite, montrons que 3⇒2. Rappelons que, par définition des HMSC causaux, tout $m \in \Gamma(\alpha) \cup \Gamma(\beta)$ est weak-FIFO, alors tout $m \in \text{MSC}(\alpha) \cup \text{MSC}(\beta)$ est weak-FIFO. Ainsi, la fonction $\eta : Lin(\mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})) \rightarrow \mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I})$ qui ordonne totalement les événements sur une même instance et qui apparie le n-ième envoi de messages avec la n-ième réception, est un morphisme. Et finalement, pour tous rationnels L et L' et tout morphisme η, $L = L'$ implique que $\eta(L) = \eta(L')$.

Enfin, montrons que 2⇒1. Supposons que, pour tout $m \in \Gamma(\alpha) \cup \Gamma(\beta)$, m est D-cohérent. Alors, tout $m \in \text{MSC}(\alpha) \cup \text{MSC}(\beta)$ est D-cohérent. Ainsi, la fonction $\eta : \mathbb{MSC}(\mathcal{P},\mathcal{M},\mathcal{I}) \rightarrow \mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)$ qui "efface", dans les MSC qui lui sont donnés en entrée, les causalités qui ne sont pas dans $I = \Sigma^2 - D$, est un morphisme. En utilisant le même résultat que pour l'implication précédente, on obtient le résultat escompté. □

En pratique, il est tout à fait possible de s'intéresser aux différentes sémantiques exprimées par une spécification, même si dans le cas général, c'est le langage de linéarisation qui semble le plus étudié et outillé (c'est l'unique sémantique que permettent de manipuler les outils classiques de model-checking). En fonction du choix de la sémantique et des restrictions syntaxiques que respectent le modèle et la formule, la proposition 3.10 nous permet de donner un indice de précision à la réponse donnée par l'algorithme de model-checking. Dans le cas où la réponse donnée est négative (c'est-à-dire si le modèle ne respecte pas la propriété exprimée par la formule logique), il est possible que cela soit dû à une mauvaise formulation de la spécification.

3.3.2 Model-checking de HMSC causaux globalement coopératifs

Le model-checking étant indécidable pour les HMSC, il est donc aussi indécidable pour les HMSC causaux. Cependant, nous montrons, dans cette partie, que les corationnels de $\mathrm{REX}(\mathbb{P}(\Sigma, D))$ correspondent à la classe syntaxique des HMSC causaux globalement coopératifs (qui étend la classe du même nom pour le HMSC). Plus précisément, lorsque l'on applique aux HMSC causaux la définition des corationnels de $\mathrm{REX}(\mathbb{P}(\Sigma, D))$, on obtient la définition suivante :

Définition 3.11 (HMSC causaux globalement coopératifs) *Soient (Σ, D), un alphabet concurrent localisé et α, un* HMSC *D-causal. On dira que α est* globalement coopératif *si, pour chaque sous-expression $\beta^\star$ de α :*

- *tout $m \in \mathcal{L}(\beta)$ a un graphe de communication connexe ;*
- *le graphe $(\Gamma(\beta), \not\parallel)$ est D-connexe.*

Nous pouvons tout d'abord remarquer que, décider si un HMSC causal est globalement coopératif, peut se faire par une procédure de décision CoNP-complète et que cette classe est exactement celle qui correspond aux corationnels lorsque l'on plonge les HMSC causaux dans le monoïde de traces à l'aide de la relation d'indépendance $\|$. Ceci nous permet d'appliquer les résultats vus précédemment aux HMSC causaux.

Proposition 3.12 *Soient (Σ, D), un alphabet concurrent et localisé, α et β, deux* HMSC *causaux et φ, une formule de logique temporelle dont les propositions atomiques sont des* HMSC *causaux premiers. Si β est globalement coopératif, alors nous pouvons construire un automate $\mathcal{B}$ tel que $\mathcal{L}_{\Gamma(\beta)^\star}(\mathcal{B}) = [\mathcal{L}_{\Gamma(\beta)^\star}(\beta)]_{\sim_\|}$ et $|\mathcal{B}| = 2^{O(|\Gamma(\beta)|\cdot|\beta|)}$. Ceci entraîne les résultats suivants :*

- *décider si $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\alpha) \cap \mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\beta) = \emptyset$ est PSPACE-complet ;*
- *décider $\mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\alpha) \subseteq \mathcal{L}_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}(\beta)$ est EXPSPACE-complet.*
- *décider si $\beta \vDash \varphi$ est PSPACE-complet si φ est une formule de $LTL_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}$;*
- *décider si $\beta \vDash \varphi$ est dans PSPACE si φ est une formule de $CTL_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}$;*
- *décider si $\beta \vDash \varphi$ est PSPACE-complet si φ est une formule de $CTL^*_{\mathbb{MSC}((\mathcal{P},\mathcal{M},\mathcal{I}),D)}$.*

Preuve: Ce résultat est immédiat, il suffit d'appliquer aux HMSC causaux les résultats montrés dans la partie précédente pour les expressions corationnelles de $\mathrm{REX}(\mathbb{P}(\Sigma, D))$. En particulier, se référer à la proposition 3.9 et au théorème 3.8. □

3.3.3 Comparaison avec les autres modèles

Les résultats présentés dans cette section sont intéressants à comparer à ceux déjà existants sur les cHMSC. En effet, il a déjà été dit que les cHMSC globalement coopératifs permettaient de générer des ensembles de MSC qui ne sont pas finiment engendrés. De plus, le model-checking est décidable sur ces modèles, avec des complexités similaires aux modèles séquentiels (en tout cas dans le cadre du model-checking négatif) et aux complexités que nous avons présentées dans ce chapitre.

Tout d'abord, dans le chapitre 2, nous avons montré que les HMSC causaux réguliers avaient un langage régulier de linéarisations. Par conséquent, les HMSC causaux réguliers sont inclus dans les cHMSC réguliers qui permettent d'engendrer *tous* les ensembles de MSC dont la linéarisation est régulière.

Ensuite, la figure 3.6 donne un HMSC causal globalement coopératif dont le langage de MSC n'est pas existentiellement borné. Par conséquent, il n'existe pas de cHMSC globalement coopératif qui génère le même langage. De plus, un HMSC causal α possédera toujours dans son langage des MSC engendrés par le HMSC construit à partir de α (en rajoutant des causalités sur chaque processus pour obtenir un ordre local). Or, il existe des cHMSC qui génèrent des langages dont aucun élément n'est finiment engendré. Par conséquent les HMSC causaux et les cHMSC sont deux modèles sont incomparables.

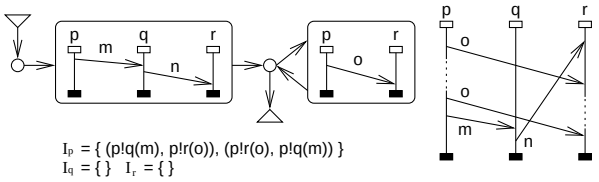


FIG. *3.6 – L'automate $\mathcal{A}_\alpha$ d'un* HMSC *causal globalement coopératif α dont le langage n'est pas existentiellement borné.*

Enfin, la figure 3.7 donne un résumé de l'inclusion de différents langages permettant d'engendrer des MSC. Le model-checking est décidable pour tous les cHMSC globalement coopératifs. Il est donc clair que nos travaux complètent le panorama existant en donnant de nouveaux outils pour vérifier des systèmes parallèles et répartis non bornés.

3.4 Conclusions et perspectives

Résumé

Dans ce chapitre, nous avons présenté une nouvelle approche permettant de faire du model-checking de modèles de systèmes parallèles et répartis, pouvant avoir un espace d'états infini. La technique présentée s'appuie sur l'utilisation de techniques classiques de model-checking appliquées à une représentation globale des interactions par des morceaux d'ordres partiels. En particulier, nous avons montré deux résultats importants.

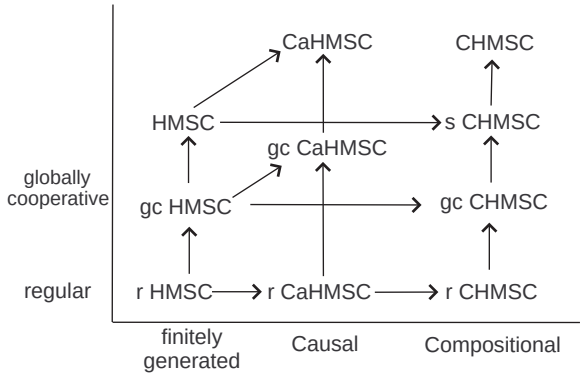


FIG. *3.7 – Comparaison des différents modèles engendrant des* MSC. *Le model-checking est décidable pour tous les globalement coopératifs.*

D'une part, nous avons montré qu'en limitant le pouvoir d'expression des langages de spécification aux expressions corationnelles de pomsets, il était possible de vérifier n'importe quelle formule d'ordres partiels, tout en gardant des complexités de résolution comparables à celles du model-checking de systèmes séquentiels. Ce résultat est, à nos yeux, très important, puisqu'il démontre que l'on peut utiliser des modèles d'ordres partiels pour gagner de la concision tout en conservant de très bonnes propriétés algorithmiques. Ce résultat montre aussi que l'on est capable de vérifier des modèles parallèles et répartis en ne connaissant pas de borne a priori sur la capacité des canaux de communication utilisés par le système sous-jacent.

D'autre part, nous avons montré qu'en limitant le pouvoir d'expression des formules, nous pouvons faire du model-checking positif et négatif pour tout type de modèle. Ce qui signifie que tous les modèles de haut niveau peuvent être analysables automatiquement, si la propriété à tester peut s'exprimer en terme de corationnel de pomsets. Encore une fois, ce résultat montre l'intérêt des modèles de haut niveau, qui permettent de décrire des comportement de systèmes non bornés, en ne perdant aucune capacité d'analyse.

Nos travaux semblent donc indiquer que la difficulté majeure se place maintenant dans la conception de modèles de haut niveau : en effet, nous avons reporté la difficulté de l'analyse à la tâche de conception. Cependant, nous pensons que le modèle des HMSC causaux globalement coopératifs, qui permet de décrire de manière concise et intuitive, à la fois des interactions par échange de messages asynchrones et des interactions par mémoire partagée, est un bon compromis : il mélange à la fois l'intuition provenant de la notation très simple des HMSC et les aspects de dépendances entre instructions qui proviennent des traces de Mazurkiewicz. Les résultats énoncés sur les langages rationnels de pomsets s'appliquent naturellement aux HMSC causaux, ce qui fait de cette classe, un bon candidat pour le compromis expressivité/décidabilité que nous recherchons dans cette thèse.

Perspectives

Tout d'abord, une première perspective serait d'appliquer les nombreux résultats obtenus sur les traces aux langages de pomsets premiers pour obtenir de nouvelles logiques pour spécifier des expressions rationnelles de pomsets ou des HMSC causaux. De même, il serait intéressant de mélanger les deux modèles de haut niveau permettant de décrire des systèmes non bornés, à savoir les HMSC causaux globalement coopératifs, que nous avons introduits dans cette thèse et les CHMSC globalement coopératifs. Cependant les techniques employées pour prouver les résultats de décidabilité du model-checking s'appuient sur des outils différents : décomposition en pomsets premiers pour l'un, utilisation d'une borne existentielle pour l'autre. Réussir à combiner ces deux approches donnerait lieu à des modèles encore plus expressifs, tout en conservant les bonnes propriétés de décision de ces deux modèles.

Ensuite, il parait important de confronter les méthodes que nous développons dans ce chapitre aux méthodes actuelles mises en œuvre dans des outils de model-checking. Bien sûr, l'intérêt de notre approche est de vérifier de manière exacte des familles de systèmes infinis, ce que les approches classiques ne sont pas capables de faire. Mais nous pensons que l'abstraction que procure la décomposition en MSC causaux premiers peut aussi améliorer grandement la concision des modèles manipulés et donc les performances de l'analyse des systèmes bornés. En particulier, l'approche qui consiste à mettre au point une sur-couche logicielle, qui réalise les actions suivantes, nous semble prometteuse :

1. transformer les problèmes que nous nous posons sur les ordres en problèmes sur des traces en utilisant la décomposition en MSC causaux premiers ;
2. faire appel à des outils classiques de model-checking tels que SPIN ou SMV pour résoudre ces problèmes ;
3. finalement, retraduire le résultat vers les MSC causaux.

Enfin, la méthodologie présentée dans ce chapitre, c'est-à-dire la recherche de sous-classes reconnaissables dans des langages de haut niveau afin d'appliquer les algorithmes classiques du model-checking utilisant des automates, peut être généralisée facilement au model-checking de n'importe quelle logique décidable (notamment au μ-calcul). De même, il est envisageable de généraliser ces techniques à des domaines différents, mais qui utilisent intensivement les automates comme techniques de preuve (jeux, etc.).

Chapitre 4

Vérification partielle de modèles

Nous avons vu, dans la partie I de ce document, différents modèles pour spécifier des systèmes parallèles et répartis et nous avons introduit un modèle mixte localement asynchrone et globalement communicant qui les généralise.

Ce chapitre constitue, avec le chapitre précédent, la partie II de cette thèse, consacrée à la *vérification* de systèmes mixtes. Plus précisément, dans ce chapitre, nous nous intéressons aux analyses de *vérification* de vues *partielles* d'un modèle. Ces techniques ont vocation à se dérouler avant l'exécution du système à vérifier. De plus, nous supposons que les propriétés à vérifier portent sur une partie, seulement, du modèle.

Plus précisément, nous nous concentrons, dans ce chapitre, sur la définition d'outils théoriques pour manipuler la projection de modèles de haut niveau.

Tout d'abord, nous définissons un nouveau modèle, appelé boxed pomset qui est à la fois proche des pomsets et qui a la propriété d'avoir une opération de projection qui laisse stable les rationnels de ce modèle.

Ensuite, le modèle des boxed pomsets nous sert à montrer deux résultats importants sur les rationnels de pomsets. D'une part, nous montrons que la projection d'un rationnel de pomsets peut être représentée à l'aide d'un générateur construit à l'aide d'une expression rationnelle de boxed pomsets. D'autre part, nous montrons que, savoir si la projection d'un rationnel de pomsets reste un rationnel de pomsets, est décidable.

Enfin, nous utilisons ce dernier résultat pour montrer que, savoir si la vue partielle d'un modèle d'ordres partiels appartient à un ensemble de comportements admissibles (c'est-à-dire le model-checking positif) est décidable.

Pour finir, la partie III de ce document sera consacrée à la supervision de systèmes parallèles et répartis mixtes.

Contexte

Dans le chapitre 3, nous avons vu que la vérification de modèle consiste, à partir d'un modèle du système et d'une formule logique exprimant des propriétés de bon comportement, à s'assurer que toutes les exécutions du modèle satisfassent la formule. C'est donc une analyse importante qui permet d'assurer certains critères de qualité à un système, avant même son exécution.

Bien que la complexité du model-checking pour les logiques temporelles classiques (telles que LTL, CTL et CTL*) soit logarithmique en la taille du modèle, en pratique, les systèmes considérés sont souvent trop grands pour être testés de manière exhaustive. Ainsi, certaines parties de ces modèles doivent être simplifiées, c'est ce qui est généralement appelé *abstraction*. L'abstraction est un point clé de tout procédé automatique de vérification de larges systèmes ([Clarke 00, Govindaraju 00, Pnueli 00]).

Dans le cadre des modèles d'ordres partiels, la nécessité de l'abstraction est aussi importante, puisque, comme nous l'avons montré dans le chapitre 3, même si la complexité reste la même pour le model-checking positif et négatif, la complexité du model-checking pour les logiques d'ordres partiels, lorsque la taille de la formule est fixée et que le modèle est corationnel, est exponentiellement plus grande que dans le cas séquentiel.

Dans la suite de ce chapitre, nous considérons comme abstraction, l'opération qui consiste à restreindre les ensembles d'événements à ceux étiquetés par un certain alphabet, comme pour les automates. Dans ce cadre, le problème est que l'on risque de perdre des causalités et donc de définir plus de comportements que ce que l'on voudrait : l'abstraction ainsi définie est une approximation. Dans le but de faire une abstraction exacte, il faut tenir compte de l'ordre partiel qui relie les événements auxquels on se restreint. La restriction qui nous intéresse alors correspond à la notion classique de projection d'ordres partiels étiquetés.

Ces méthodes de projection permettent de se focaliser sur les interactions entre certaines entités spécifiques, en projetant uniquement sur les actions qui sont localisées sur ces entités. Elles permettent aussi de comparer des scénarios qui décrivent des interactions différentes (mais non disjointes) en se focalisant sur les éléments communs. Ainsi, étant donné une formule logique définie sur un alphabet Σ_o, nous nous intéressons au model-checking partiel qui consiste à projeter le modèle sur Σ_o et à vérifier que cette projection vérifie la formule donnée.

Dans ce chapitre, nous considérons donc que nous avons un modèle, donné sous la forme d'une expression rationnelle de pomsets étiquetés par Σ et une formule logique d'ordres partiels, définie sur des pomsets étiquetés par $\Sigma_o \subseteq \Sigma$. Nous cherchons à répondre à la question suivante : est-ce que la projection du langage du modèle peut être exprimée à l'aide d'une expression rationnelle de pomsets. Dans le cas positif, les techniques que nous avons développées dans le chapitre précédent s'appliquent encore. Dans le cas négatif, le langage obtenu n'est pas finiment engendré et il n'est, a priori, pas possible d'appliquer les techniques de vérification développées dans cette thèse sur ce modèle, sauf dans le cadre du model-checking positif (qui consiste à vérifier que la projection du modèle reste dans un ensemble de comportements bien définis).

Des travaux similaires ont été menés par Genest, Hélouët et Muscholl dans [Genest 03], sur la projection de HMSC. Plus précisément, ils se sont intéressés à caractériser une classe de CHMSC dont l'existence d'une représentation par un HMSC est décidable. Or, cette classe de CHMSC correspond exactement à la projection des HMSC. Ils ont donc montré qu'il était décidable (en espace polynomial) de savoir si la projection d'un HMSC peut se représenter à l'aide d'un HMSC. Les méthodes développées dans ces travaux ne sont pas applicables aux expressions rationnelles de pomsets, car elles s'appuient sur une borne existentielle et une traduction vers les automates communicants, ce qui n'a pas de sens pour les expressions rationnelles de pomsets.

Organisation du chapitre

Ce chapitre est découpé comme suit. La partie 4.1 introduit un nouveau modèle, celui des *boxed pomsets*. Nous montrons, dans la partie 4.2, que ce modèle possède de bonnes propriétés lorsqu'il est utilisé avec des projections : en effet, les ensembles rationnels de boxed pomsets sont stables par projection, ce qui nous permet de construire un générateur rationnel fini pour représenter la projection du langage de n'importe quelle expression rationnelle de pomsets. Ensuite, la partie 4.3 montre que l'on peut utiliser ce nouveau modèle pour décider quand la projection d'un langage rationnel de pomsets reste un langage rationnel de pomsets. Enfin, la partie 4.4 conclut ce chapitre et donne quelques perspectives liées aux travaux que nous avons menés.

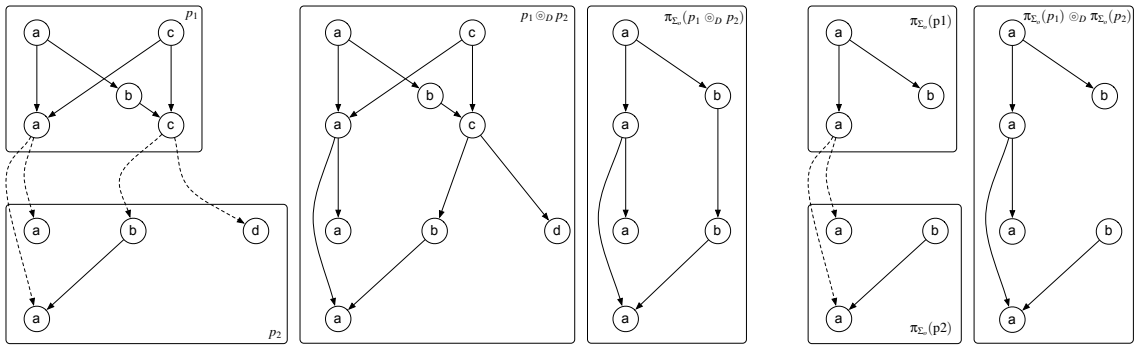
4.1 Boxed pomsets

Nous introduisons, dans cette partie, un nouveau modèle fondé sur les pomsets et dont les ensembles rationnels restent stables par projection, contrairement à ceux des pomsets. Pour ce nouveau modèle, la composition et la projection sont légèrement différentes de celles des pomsets : sont conservés des événements supplémentaires qui permettent de retrouver les causalités perdues lors de la projection. Dans la partie 4.1.1, nous commençons par donner un exemple de langage rationnel qui n'est pas stable par projection. Puis, dans la partie 4.1.2, nous donnons la définition des boxed pomsets, ainsi que les opérations de composition et de projection associées.

4.1.1 Pomsets et projection

Nous commençons la partie technique de ce chapitre en donnant un exemple de composition de pomsets et de projection. La figure 4.1 montre que la projection de la composition de deux pomsets n'est, en général, pas la composition de la projection de ces deux pomsets. En effet, pour $D = \{(a,a),(c,b),(c,d)\}$ et $\Sigma_o = \{a,b\}$, nous avons $\pi_{\Sigma_o}(p_1) \circledcirc_D \pi_{\Sigma_o}(p_2)$ qui n'est pas isomorphe à $\pi_{\Sigma_o}(p_1 \circledcirc_D p_2)$, car la causalité entre les deux événements étiquetés par b est perdue dans le second cas. De plus, à cause de cette causalité, il n'existe pas de décomposition possible de $\pi_{\Sigma_o}(p_1 \circledcirc_D p_2)$ en deux pomsets non nuls.

Nous pouvons maintenant généraliser ces résultats aux expressions rationnelles de pomsets. Il suffit, en effet, de considérer la projection sur Σ_o du langage de l'expression rationnelle



(a) composition/projection (b) projection/composition

FIG. *4.1 – Composition de pomsets et projection, avec* $D = \{(a,a),(c,b),(c,d)\}$ *et* $\Sigma_o = \{a,b\}$.

$p_1 a^\star p_2$. Les éléments de cette projection sont de la même forme que $\pi_{\Sigma_o}(p_1 \circledcirc_D p_2)$ de la figure 4.1, mais avec un nombre non borné d'événements étiquetés par a qui succèdent à ceux de p_1 et qui précèdent ceux de p_2. De la même manière que pour $\pi_{\Sigma_o}(p_1 \circledcirc_D p_2)$, ces éléments ont une causalité qui relie les deux événements étiquetés par b. Ils ne sont donc pas décomposables en un produit de pomsets non nuls plus petits. Par conséquent, la projection du langage de $p_1 a^\star p_2$ n'est pas finiment engendrée et n'est donc pas le langage d'une expression rationnelle de pomsets.

De manière intuitive, le modèle des boxed pomsets que nous définissons dans ce chapitre va rajouter des informations aux pomsets, pour se souvenir des causalités qu'efface la projection. Ces informations supplémentaires consistent en deux ports, comportant respectivement, pour chaque action de Σ, les événements minimaux et maximaux du pomset considéré. Ensuite, nous définissons une opération de projection qui ne modifie pas les ports et conserve les causalités liées aux événements de ces ports et une opération de composition distributive sur celle de la projection. Enfin, nous définissons des opérateurs d'encapsulation, qui permettent, à partir d'un pomset, de rajouter les ports pour obtenir un boxed pomset et inversement, une opération de désencapsulation qui, à partir d'un boxed pomset, oublie les ports pour obtenir un pomset.

4.1.2 Définition des boxed pomsets

Afin de pouvoir travailler plus facilement avec la projection et la composition de pomsets, nous introduisons dans ce chapitre un nouveau modèle, appelé *boxed pomsets* que nous avons publié dans [Gazagnaire 07c]. Avant d'introduire ce modèle, nous donnons quelques définitions préliminaires.

Tout d'abord, fixons un ensemble d'actions Σ. Un *port* est un ensemble d'événements étiquetés (E,λ), avec $\lambda : E \to \Sigma$ et qui ne comporte pas de répétitions d'étiquettes, c'est-à-dire que pour tout lettre a de Σ, $|\lambda^{-1}(a) \cap E| = 1$.

Ensuite, nous dirons qu'une relation de causalité $\leq$ *branche* un ensemble d'événements étiquetés (E_1,λ_1) à un autre ensemble d'événements étiquetés (E_2,λ_2), lorsque, pour toute

lettre a, tous les événements étiquetés par a dans E_1 précèdent causalement tous les événements étiquetés par a dans E_2. Plus formellement, lorsque les fonctions d'étiquetage ne sont pas ambigües, nous notons $\leq_{E_1 \rightsquigarrow E_2} = \{(e_1, e_2) \in E_1 \times E_2 \mid \lambda_1(e_1) = \lambda_2(e_2)\}$ et nous dirons que $\leq$ branche E_1 à E_2 si, et seulement si, $\leq_{E_1 \rightsquigarrow E_2} \subseteq \leq$.

Définition 4.1 (boxed pomset) *Un* boxed pomset *est la classe d'isomorphisme* $[E^- \uplus E \uplus E^+, \leq, \lambda]$ *d'une structure* $(E^- \uplus E \uplus E^+, \leq, \lambda)$*, où :*

- (E^-, λ) *est un port, appelé* port d'entrée *;*
- (E^+, λ) *est un port, appelé* port de sortie *;*
- E *est un ensemble d'événements appelé* boîte interne *;*
- $\leq \subseteq (E^- \uplus E) \times (E \uplus E^+)$ *est une relation d'ordre partiel qui vérifie* $\leq_{E^- \rightsquigarrow (E \uplus E^+)} \subseteq \leq$ *et* $\leq_{(E^- \uplus E) \rightsquigarrow E^+} \subseteq \leq$*, c'est-à-dire* $\leq$ *branche* E^- *à* $(E \uplus E^+)$ *et* $(E^- \uplus E)$ *à* E^+ *;*
- $\lambda : E^- \uplus E \uplus E^+ \to \Sigma$ *est une fonction d'étiquetage.*

Un boxed pomset b peut être vu comme un pomset que l'on a encapsulé, avec des accès, pour chaque lettre de Σ, aux événements minimaux et maximaux étiquetés par cette lettre. Ainsi, afin de faciliter les notations, nous écrirons le boxed pomset b en nommant explicitement ses ports, c'est-à-dire sous la forme d'une structure $[E_b^- \uplus E_b \uplus E_b^+, \leq_b, \lambda_b]$.

La figure 4.2 montre un exemple de deux boxed pomsets isomorphes, avec $\Sigma = \{a, b, c, d, e\}$. Ils sont représentés par des pomsets pour lesquels nous distinguons explicitement, à l'aide de rectangles distincts, le port d'entrée, la boîte interne et le port de sortie. Sur les figures, le port d'entrée sera toujours représenté en haut, le port de sortie, en bas. De plus, pour des raisons de lisibilité, il peut arriver que nous ne représentions pas les éléments des ports d'entrée et de sortie dont l'étiquette n'apparaît pas dans la boîte interne et dont la causalité est uniquement due au branchement, ce qui est le cas, dans la figure 4.2, pour les événements étiquetés par e.

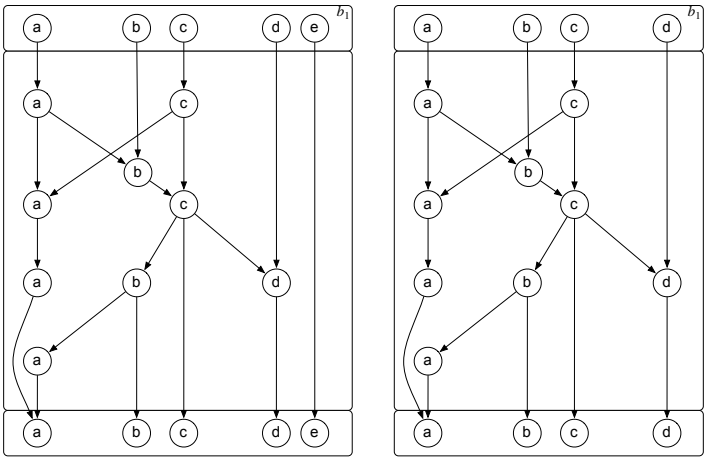


FIG. 4.2 – *Deux boxed pomsets isomorphes, avec* $\Sigma = \{a, b, c, d, e\}$.

Projection de boxed pomsets

Nous introduisons maintenant l'opération de projection adaptée à ce modèle. Cette opération conserve les informations de causalité liées aux ports. Nous verrons par la suite, que cette information est suffisante pour pouvoir définir une composition telle que la projection soit distributive sur celle-ci.

Définition 4.2 (projection) *La* projection *d'un boxed pomset b sur un alphabet observable Σ_o est une fonction $\widehat{\pi}_{\Sigma_o}$ qui restreint la boîte interne de b aux événements qui sont étiquetés par Σ_o, sans modifier les ports d'entrée et de sortie, c'est-à-dire :*

$$\widehat{\pi}_{\Sigma_o}(b) = [E_b^- \uplus (E_b \cap E_{\Sigma_o}) \uplus E_b^+, \leq_b \cap (E'_{\Sigma_o})^2, \lambda_b]$$

avec $E_{\Sigma_o} = \lambda^{-1}(\Sigma_o)$ l'ensemble des événements observables et $E'_{\Sigma_o} = E_b^- \uplus E_{\Sigma_o} \uplus E_b^+$ l'union des événements observables et des événements qui appartiennent à un port.

La définition précédente montre l'utilité des ports : ils permettent de conserver des informations de causalité (entre les ports et la boîte interne, ou entre les ports) malgré la projection des événements de la boîte interne.

Cette définition est illustrée par la figure 4.3, avec $\Sigma_o = \{a, b\}$. Sur cette figure, nous pouvons voir que les événements étiquetés par c et d ont été effacés dans la boîte interne, mais les événements minimaux et maximaux correspondants sont toujours présents dans les ports d'entrée et de sortie, ainsi que les causalités associées.

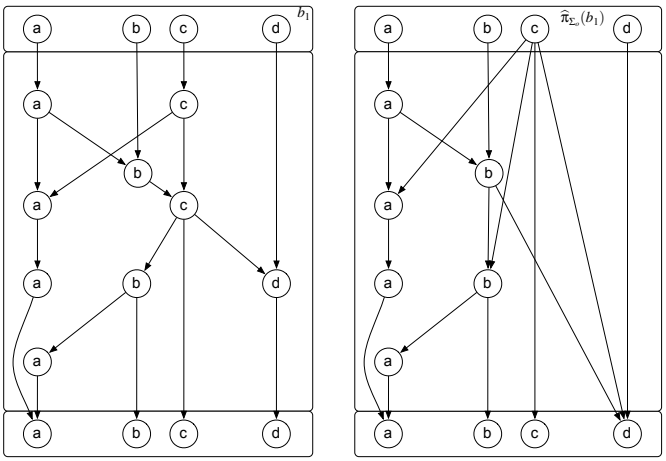


FIG. *4.3 – Projection d'un boxed pomset, avec* $\Sigma_o = \{a, b\}$.

Composition de boxed pomsets

Nous allons maintenant étendre la composition, définie sur les pomsets, aux boxed pomsets. Cette composition est une opération stable : elle ne modifie pas la structure globale d'un boxed pomset. De manière intuitive, la composition de deux boxed pomsets b_1 et b_2, notée $b_1 \boxdot_D b_2$, effectue la composition des ports intermédiaires (le port de sortie de b_1 et

celui d'entrée de b_2) puis conserve la causalité ajoutée, tout en "effaçant" les événements de ces ports. De plus, les ports d'entrée et de sortie sont ensuite modifiés pour qu'ils correspondent, respectivement, aux événements minimaux et maximaux de la structure obtenue. Plus formellement :

Définition 4.3 (composition) *Soient* $b_i = [E_i^- \uplus E_i \uplus E_i^+, \leq_i, \lambda_i]$, *pour* $i \in \{1,2\}$, *deux boxed pomsets étiquetés par* Σ *et* $D \subseteq \Sigma^2$. *La* composition *de* b_1 *et* b_2 *est définie par* $b_1 \boxdot_D b_2 = [E^- \uplus E \uplus E^+, \leq, \lambda]$, *la classe d'isomorphisme d'une structure* $(E^- \uplus E \uplus E^+, \leq, \lambda)$, *où :*

- (E^-, λ) *est un port, auquel* E_1^- *et* E_2^- *sont branchés ;*
- E *est isomorphe à l'union des boîtes internes* $E_1 \uplus E_2$ *;*
- (E^+, λ) *est un port, qui est branché à* E_1^+ *et* E_2^+ *;*
- $\leq = (\leq_1 \cup \leq_2 \cup \leq_D \cup \leq_{E^- \rightsquigarrow (E_1^- \uplus E_2^-)} \cup \leq_{(E_1^+ \uplus E_2^+) \rightsquigarrow E^+})^* \cap (E^- \uplus E \uplus E^+)^2$ *est la restriction, aux événements de* $E^- \uplus E \uplus E^+$, *de la fermeture transitive de l'union des relations d'ordres partiels de* b_1, *de* b_2, *de la composition des ports intermédiaires de* b_1 *et* b_2 *:* $\leq_D = \{(e, e') \in E_1^+ \times E_2^- \mid (\lambda(e), \lambda(e')) \in D\}$ *et des relations de branchement entre les ports ;*
- $\lambda = \lambda_1 \cup \lambda_2$ *est l'union des fonctions d'étiquetage de* b_1 *et de* b_2.

Les événements des ports d'entrée et de sortie de $b_1 \boxdot_D b_2$ sont branchés de telle manière qu'ils deviennent les événements minimaux (pour ceux du port d'entrée) et maximaux (pour ceux du port de sortie) du boxed pomset produit. La boîte interne de $b_1 \boxdot_D b_2$ est isomorphe à l'union des boîtes internes de b_1 et b_2 à laquelle on a rajouté les causalités induites par la composition (et l'effacement) des ports intermédiaires. De plus, la définition assure que les ports et la boîte interne sont branchés correctement. Enfin, nous notons ε l'élément neutre de $\boxdot_D$, qui est le boxed pomset dont la boîte interne est vide et dont les ports ne sont reliés que par des causalités induites par le branchement. Muni de la loi de composition et de l'élément neutre définis précédemment, l'ensemble des boxed pomset $(\mathbb{B}(\Sigma, D), \boxdot_D, \varepsilon)$ forme un monoïde que nous noterons simplement $\mathbb{B}(\Sigma, D)$.

La figure 4.4 donne un exemple de composition de boxed pomsets, avec $b_4 = b_2 \boxdot_D b_3$ où $D = \{(a,a); (c,b); (c,d)\}$. Les relations de causalités ajoutées sont représentées par des lignes en pointillé et les nouveaux ports d'entrée et de sortie sont représentés par les rectangles en haut et en bas des deux boxed pomsets composés.

Traductions entre pomsets et boxed pomsets

Nous définissons maintenant des opérateurs qui permettent de passer du monoïde des pomsets à celui des boxed pomsets. L'idée principale qui motive le passage d'un monoïde à un autre est que les problèmes liés à la projection sont beaucoup plus faciles à résoudre dans $\mathbb{B}(\Sigma, D)$ que dans $\mathbb{P}(\Sigma, D)$.

L'opérateur *d'encapsulation* $B : \mathbb{P}(\Sigma, D) \to \mathbb{B}(\Sigma, D)$ (pour "Boxing operator") est utilisé pour construire un boxed pomset $B(p)$ à partir d'un pomset p. Le boxed pomset construit possède une boîte interne qui correspond exactement à p et des ports d'entrée et de sortie branchés de manière adéquate. Plus formellement, $B(p)$ est défini comme :

$$B(p) = [E^- \uplus E_p \uplus E^+, (\leq_p \cup \leq_{E^- \rightsquigarrow E_p} \cup \leq_{E_p \rightsquigarrow E^+})^*, \lambda^- \cup \lambda \cup \lambda^+]$$

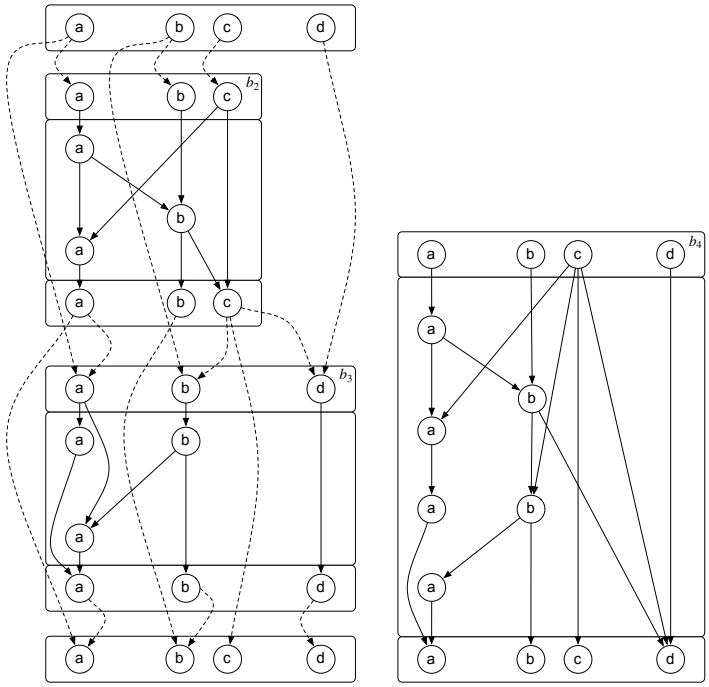


FIG. *4.4 – Composition de boxed pomsets, avec* $b_2 \boxdot_D b_3 = b_4$ *et* $D = \{(a,a),(c,b),(c,d)\}$.

où (E^-, λ^-) et (E^+, λ^+) sont des ports. Cette définition s'étend naturellement aux expressions rationnelles. Etant donné une expression rationnelle de pomsets α, $B(\alpha)$ est l'expression rationnelle dans laquelle les pomsets générateurs $p \in \Gamma(\alpha)$ sont remplacés par $B(p)$. Par exemple, $B(ab^\star) = B(a)B(b)^\star$.

De même, l'opérateur de *désencapsulation* $U : \mathbb{B}(\Sigma, D) \to \mathbb{P}(\Sigma, D)$ est utilisé pour extraire d'un boxed pomset, sa boîte interne :

$$U(b) = [E_b, \leq_b \cap E_b^2, \lambda]$$

De la même façon que pour l'opérateur d'encapsulation, l'opérateur de désencapsulation s'étend aux expressions rationnelles de boxed pomsets en remplaçant les boxed pomsets générateurs par leur désencapsulation. Par exemple, $U(a^\star + b) = U(a)^\star + U(b)$.

Pour illustrer ces définitions, considérons la figure 4.5. Celle-ci montre, à gauche, l'encapsulation du pomset p_1 de la figure 4.1. De même, elle illustre, à droite, la désencapsulation du boxed pomset b_2 de la figure 4.4. Cet exemple montre que U n'est pas injectif, car plusieurs boxed pomsets peuvent avoir la même désencapsulation que $U(b_2)$: il suffit d'ajouter des causalités entre les ports et la boîte interne de b_2.

Dans la partie suivante, nous étudions les propriétés des boxed pomsets. En particulier, nous montrons que ce modèle est stable par projection et qu'il peut être utilisé pour caractériser la projection d'une expression rationnelle de pomsets.

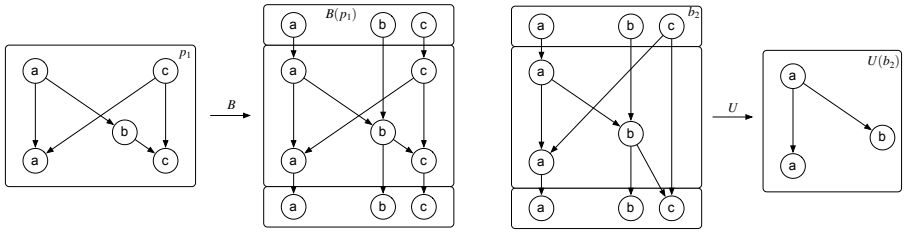


FIG. *4.5 – Opérateurs d'encapsulation et de désencapsulation.*

4.2 Propiétés des boxed pomsets

Cette partie introduit les résultats principaux concernant les boxed pomsets. Tout d'abord, dans la partie 4.2.1, nous donnons les propriétés des modèles finis qui relient les opérateurs définis dans la partie précédente et les opérateurs de projection et de composition. Dans la partie 4.2.2, nous étendons ces résultats aux langages de boxed pomsets.

4.2.1 Propriétés des modèles finis

Cette partie contient trois propositions techniques qui nous donnent des outils de base pour manipuler les boxed pomsets. Plus précisément :

1. La proposition 4.4 indique que l'opérateur d'encapsulation est l'inverse à gauche de l'opérateur de désencapsulation.
2. La proposition 4.5 indique que les opérateurs de désencapsulation et de projection commutent.
3. La proposition 4.6 indique que l'opérateur d'encapsulation est un morphisme de monoïde.

Encapsulation et désencapsulation

Premièrement, nous pouvons remarquer, en observant la figure 4.5, que $U(B(p_1)) = p_1$. La proposition 4.4 démontre que ce résultat peut se généraliser, car l'opérateur d'encapsulation est l'inverse à droite de l'opérateur de désencapsulation. Cependant, il n'en est pas l'inverse à gauche, car l'opérateur de désencapsulation n'est pas injectif (il suffit, en effet, d'observer que $B(U(b_2)) \neq b_2$, dans cette même figure 4.5).

Proposition 4.4 *Soit $p \in \mathbb{P}(\Sigma, D)$, un pomset. Alors, $U(B(p)) = p$.*

Preuve: Rappelons simplement que, par définition :

$$B(p) = [E^- \uplus E_p \uplus E^+, (\leq_p \cup \leq_{E^- \rightsquigarrow E_p} \cup \leq_{E_p \rightsquigarrow E^+})^*, \lambda_{B(p)}]$$

Or, remarquons que $(\leq_p \cup \leq_{E^- \rightsquigarrow E_p} \cup \leq_{E_p \rightsquigarrow E^+})^* \cap E_p^2 = \leq_p$ et que $\lambda_{B(p)}$ coïncide avec λ_p sur E_p. Nous pouvons donc en conclure que $U(B(p)) = [E_p, \leq_p, \lambda_p] = p$. □

Désencapsulation et projection

Deuxièmement, la projection et la désencapsulation sont des opérateurs qui peuvent commuter. Cette propriété s'illustre sur la figure 4.5, en considérant $b = B(p_1)$ et $\Sigma_o = \{a, b\}$. Remarquons alors que $\pi_{\Sigma_o}(U(b)) = U(b_2) = U(\widehat{\pi}_{\Sigma_o}(b))$. Plus généralement, nous avons la proposition suivante :

Proposition 4.5 *Soient $b \in \mathbb{B}(\Sigma, D)$ et $\Sigma_o \subseteq \Sigma$. Alors, $\pi_{\Sigma_o}(U(b)) = U(\widehat{\pi}_{\Sigma_o}(b))$.*

Preuve: Tout d'abord, rappelons que $U(b) = [E_b, (\leq_b \cap E_b^2), \lambda_b]$. En appliquant la définition de la projection et en notant $E_{\Sigma_o} = \lambda^{-1}(\Sigma_o)$ l'ensemble des événements observables, nous avons donc :

$$\pi_{\Sigma_o}(U(b)) = [(E_b \cap E_{\Sigma_o}), (\leq_b \cap E_b^2 \cap E_{\Sigma_o}^2), \lambda_b]$$

Ensuite, par définition : $\widehat{\pi}_{\Sigma_o}(b) = [E_b^- \uplus (E_b \cap E_{\Sigma_o}) \uplus E_b^+, (\leq_b \cap E_{\Sigma_o}'^2), \lambda_b]$, avec E'_{Σ_o} l'ensemble $E_b^- \uplus E_{\Sigma_o} \uplus E_b^+$ des événements qui sont, soit observables, soit qui apparaissent dans un port. En appliquant l'opérateur de désencapsulation et en notant λ'_b la restriction de λ_b à E_b, nous obtenons :

$$U(\widehat{\pi}_{\Sigma_o}(b)) = [(E_b \cap E_{\Sigma_o}), (\leq_b \cap E_{\Sigma_o}'^2 \cap E_b^2), \lambda'_b]$$

Pour conclure la preuve, remarquons finalement que $E'_{\Sigma_o} \cap E_b = E_{\Sigma_o} \cap E_b$. □

Encapsulation et composition

Troisièmement, la composition de pomsets et de boxed pomsets est aussi fortement liée. En effet, même si, en général, la désencapsulation n'est pas distributive sur la composition (il est facile de trouver un contre-exemple où $U(b_1) \circledcirc_D U(b_2)$ est différent de $U(b_1 \boxdot_D b_2)$, voir par exemple la figure 4.4 où $U(b_2) \circledcirc_D U(b_3) \neq U(b_4)$), l'encapsulation est distributive sur la composition. Ainsi, B est un morphisme de monoïde.

Proposition 4.6 *Soient p_1 et p_2, deux pomsets dans $\mathbb{P}(\Sigma, D)$. Alors, $B(p_1 \circledcirc_D p_2) = B(p_1) \boxdot_D B(p_2)$.*

Preuve: Commençons par fixer quelques notations. Posons $p_1 = [E_1, \leq_1]$, $p_2 = [E_2, \leq_2]$, $p_1 \circledcirc_D p_2 = [E_{1\circledcirc 2}, \leq_{1\circledcirc 2}]$, $\lambda(E_i) = \lambda(E_i^+) = \lambda(E_i^-)$ et λ l'union de toutes les fonctions d'étiquetage des structures utilisées dans cette preuve. Nous pouvons alors caractériser, avec ces notations, l'encapsulation de p_i :

$$B(p_i) = [E_i^- \uplus E_i \uplus E_i^+, (\leq_i \cup \leq_{E_i^- \rightsquigarrow E_i} \cup \leq_{E_i \rightsquigarrow E_i^+})^*, \lambda]$$

De même, nous pouvons caractériser l'encapsulation de la concaténation de p_1 et p_2.

$$B(p_1 \circledcirc_D p_2) = [E_{1\circledcirc 2}^- \uplus E_{1\circledcirc 2} \uplus E_{1\circledcirc 2}^+, (\leq_{1\circledcirc 2} \cup \leq_{E_{1\circledcirc 2}^- \rightsquigarrow E_{1\circledcirc 2}} \cup \leq_{E_{1\circledcirc 2} \rightsquigarrow E_{1\circledcirc 2}^+})^*, \lambda]$$

L'objectif de cette preuve est donc de montrer que la concaténation de $B(p_1)$ et $B(p_2)$ est bien égale au résultat précédent. Pour cela, notons la définition de cette concaténation de la manière suivante :

$$B(p_1) \boxdot_D B(p_2) = [E_{1\boxdot 2}^- \uplus E_{1\boxdot 2} \uplus E_{1\boxdot 2}^+, \leq_{1\boxdot 2}, \lambda]$$

Il faut maintenant montrer, d'une part, que $E^-_{1\circledcirc 2} \uplus E_{1\circledcirc 2} \uplus E^+_{1\circledcirc 2}$ est isomorphe à $E^-_{1\boxdot 2} \uplus E_{1\boxdot 2} \uplus E^+_{1\boxdot 2}$. Ceci est immédiat car les ports sont, par définition, isomorphes et $E_{1\circledcirc 2} = E_1 \cup E_2 = E_{1\boxdot 2}$.

D'autre part, il faut montrer que $(\leq_{1\circledcirc 2} \cup \leq_{E^-_{1\circledcirc 2} \rightsquigarrow E_{1\circledcirc 2}} \cup \leq_{E_{1\circledcirc 2} \rightsquigarrow E^+_{1\circledcirc 2}})^*$ définit la même relation d'ordre que $\leq_{1\boxdot 2}$. Il faut pour cela remarquer que, par définition, $\leq_{1\boxdot 2}$ peut s'écrire comme la restriction à $E^-_{1\boxdot 2} \uplus E_{1\boxdot 2} \uplus E^+_{1\boxdot 2}$ de la fermeture transitive de l'union des ensembles suivants :

- l'ordre induit par b_1 : $\leq_{b_1} = (\leq_1 \cup \leq_{E_1^- \rightsquigarrow E_1} \cup \leq_{E_1 \rightsquigarrow E_1^+})^*$;
- l'ordre induit par b_2 : $\leq_{b_2} = (\leq_2 \cup \leq_{E_2^- \rightsquigarrow E_2} \cup \leq_{E_2^- \rightsquigarrow E_2^+})^*$;
- l'ordre induit par la composition des ports intermédiaires : $\leq_D = \{(e, e') \in E_1^- \times E_2^+ \mid (\lambda(e), \lambda(e')) \in D\}$;
- l'ordre induit par le branchement du port d'entrée : $\leq_{E^-_{1\boxdot 2} \rightsquigarrow (E_1^- \cup E_2^-)}$;
- l'ordre induit par le branchement du port de sortie : $\leq_{(E_1^+ \cup E_2^+) \rightsquigarrow E^+_{1\boxdot 2}}$.

En posant $\leq'_D = \{(e, e') \in E_1 \times E_2 \mid (\lambda(e), \lambda(e')) \in D\}$, nous pouvons ensuite remarquer que $\leq_{1\circledcirc 2} = (\leq_1 \cup \leq_2 \cup \leq'_D)^*$, et donc que $\leq_{1\circledcirc 2} = (\leq_{b_1} \cup \leq_{b_2} \cup \leq_D)^* \cap E^2_{1\boxdot 2}$.

Ensuite, l'opérateur d'encapsulation n'ajoute pas de causalités entre les ports et la boîte interne autres que celles de branchement. Par conséquent, nous obtenons les deux égalités suivantes :

$$(\leq_{E^-_{1\boxdot 2} \rightsquigarrow (E_1^- \cup E_2^-)} \cup \leq_{E_1^- \rightsquigarrow E_1} \cup \leq_{E_2^- \rightsquigarrow E_2}) \cap (E^-_{1\boxdot 2} \cup E_{1\boxdot 2})^2 = \leq_{E^-_{1\circledcirc 2} \rightsquigarrow E_{1\circledcirc 2}}$$

$$(\leq_{E_1 \rightsquigarrow E_1^+} \cup \leq_{E_2 \rightsquigarrow E_2^+} \cup \leq_{(E_1^+ \cup E_2^+) \rightsquigarrow E^+_{1\boxdot 2}}) \cap (E_{1\boxdot 2} \cup E^+_{1\boxdot 2})^2 = \leq_{E_{1\circledcirc 2} \rightsquigarrow E^+_{1\circledcirc 2}}$$

Remarquons enfin que : $\leq_{1\circledcirc 2} \cap (E^-_{1\boxdot 2} \cap E^+_{1\boxdot 2})^2 = \emptyset$, $\leq_{E^-_{1\circledcirc 2} \rightsquigarrow E_{1\circledcirc 2}} \cap (E^+_{1\boxdot 2})^2 = \emptyset$ et $\leq_{E_{1\circledcirc 2} \rightsquigarrow E^+_{1\circledcirc 2}} \cap (E^-_{1\boxdot 2})^2 = \emptyset$.

Ce qui nous permet de conclure cette preuve en obtenant l'égalité suivante :

$$\leq_{1\boxdot 2} = (\leq_{1\circledcirc 2} \cup \leq_{E^-_{1\circledcirc 2} \rightsquigarrow E_{1\circledcirc 2}} \cup \leq_{E_{1\circledcirc 2} \rightsquigarrow E^+_{1\circledcirc 2}})^*$$

□

La figure 4.6 illustre cette proposition. A gauche de celle-ci, est représentée la composition de l'encapsulation des pomsets p_1 et p_2, avec $D = \{(a, a), (c, b), (c, d)\}$. A droite de celle-ci, est représentée l'encapsulation de la composition des pomsets p_1 et p_2, avec la même relation de dépendance. Finalement, au milieu de cette figure est représenté le résultat de ces deux opérations qui est le boxed pomset b_1, défini à la figure 4.2.

Les propositions précédentes nous donnent des outils de base pour manipuler des boxed pomsets avec les opérateurs de projection ou de composition. En particulier, ils nous permettent de reformuler la définition de la composition de boxed pomsets : pour tout $b_1, b_2 \in \mathbb{B}(\Sigma, D)$, $b_1 \boxdot_D b_2$ peut se définir comme l'encapsulation de la restriction, aux événements de l'union de $U(b_1)$ et de $U(b_2)$, de $b_1 \circledcirc_D b_2$ (les boxed pomsets pouvant être interprétés comme des pomsets). C'est à dire :

$$b_1 \boxdot_D b_2 = B(b_1 \circledcirc_D b_2)_{|U(b_1) \cup U(b_2)}$$

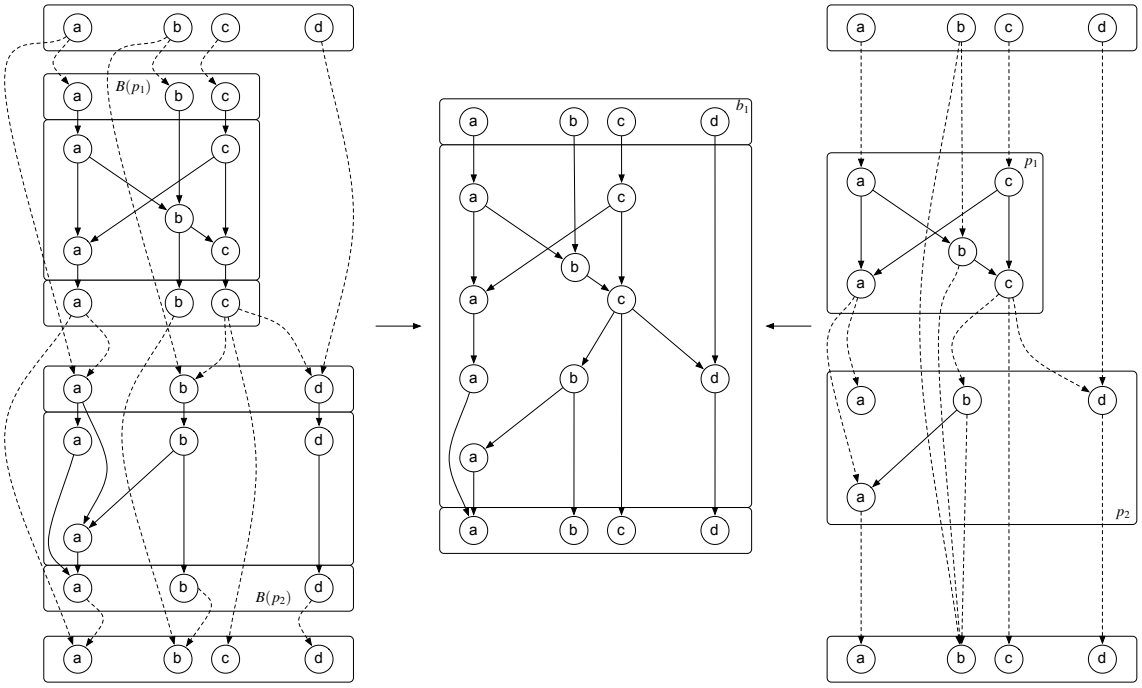


FIG. *4.6 – La composition de pomsets encapsulés est l'encapsulation de la composition de ces pomsets.*

Nous allons voir dans la partie suivante que ces bonnes propriétés peuvent s'étendre aux langages rationnels de boxed pomsets.

4.2.2 Propriétés des expressions rationnelles

Nous donnons, dans cette partie, les résultats concernant des langages rationnels de boxed pomsets et de pomsets. Ces résultats s'organisent en deux parties.

1. Nous montrons que, pour les boxed pomsets, l'opérateur de projection est distributif sur celui de la composition (proposition 4.7) et donc que $\widehat{\pi}_{\Sigma_o}$ est un morphisme de monoïde. Puis nous utilisons ce résultat pour montrer que les langages rationnels des boxed pomsets sont stables par projection (théorème 4.8).
2. Nous montrons que les langages rationnels de pomsets sont stables par encapsulation (proposition 4.9), ce qui permet de montrer le résultat principal de cette partie, à savoir la construction d'un générateur fini pour la projection de n'importe quel langage rationnel de pomsets (théorème 4.10).

Expressions rationnelles de boxed pomsets

D'une part, nous montrons que les rationnels de $\mathbb{B}(\Sigma, D)$ sont stables par projection, contrairement aux rationnels de $\mathbb{P}(\Sigma, D)$ qui ne le sont pas (voir la partie 4.1.1).

Pour arriver à montrer la stabilité de la projection, nous démontrons la proposition 4.7 qui indique que l'opérateur de projection est distributif sur celui de la composition, c'est-à-dire

que la projection de la composition de deux boxed pomsets est exactement la composition de la projection de ces mêmes boxed pomsets. Ce résultat indique, donc, que l'opérateur $\widehat{\pi}_{\Sigma_o}$ est un morphisme de monoïde.

Proposition 4.7 *Soient b_1 et b_2, deux boxed pomsets dans $\mathbb{B}(\Sigma, D)$ et $\Sigma_o \subseteq \Sigma$. Alors :*

$$\widehat{\pi}_{\Sigma_o}(b_1 \boxdot_D b_2) = (\widehat{\pi}_{\Sigma_o}(b_1)) \boxdot_D (\widehat{\pi}_{\Sigma_o}(b_2))$$

Preuve: Notons $b_i = [E_i^- \cup E_i \cup E_i^+, \leq_i]$, pour $i \in \{1,2,3\}$ et avec $b_3 = b_1 \boxdot_D b_2$. Posons ensuite $E_{\Sigma_o} = \lambda^{-1}(\Sigma_o)$ et $E_{i\Sigma_o} = E_i^- \cup E_{\Sigma_o} \cup E_i^+$. Nous avons donc, tout d'abord, par définition de la projection :

$$\widehat{\pi}_{\Sigma_o}(b_{i)} = [E_i^- \uplus (E_i \cap E_{\Sigma_o}) \uplus E_i^+, \leq_i \cap E^2_{i\Sigma_o}, \lambda_i]$$

Ensuite, considérons le produit des projections, c'est-à-dire le terme :

$$\widehat{\pi}_{\Sigma_o}(b_1) \boxdot_D \widehat{\pi}_{\Sigma_o}(b_2) = [E^-_{1\boxdot 2} \cup E_{1\boxdot 2} \cup E^+_{1\boxdot 2}, \leq_{1\boxdot 2}, \lambda_{1\boxdot 2}]$$

Montrons que les composants de ce terme sont identifiables à ceux de $\widehat{\pi}_{\Sigma_o}(b_3)$. Premièrement, il est immédiat de voir que $E^-_{1\boxdot 2}$, E_3^-, $E^+_{1\boxdot 2}$ and E_3^+ sont isomorphes. De plus, $E_{1\boxdot 2} = (E_1 \cap E_{\Sigma_o}) \uplus (E_2 \cap E_{\Sigma_o}) = E_3 \cap E_{\Sigma_o}$. Ensuite, il est tout aussi immédiat de voir que $\lambda_{1\boxdot 2} = \lambda_3$. Finalement, montrons que $\leq_{1\boxdot 2}$ et $(\leq_3 \cap E_{3\Sigma_o})$ définissent le même ordre partiel. Tout d'abord, $\leq_{1\boxdot 2}$ peut se décomposer en la restriction aux événements de $E^-_{1\boxdot 2} \cup E_{1\boxdot 2} \cup E^+_{1\boxdot 2}$ de fermeture transitive de l'union de différents ensembles :

- l'ordre induit par la projection de b_1 : $\leq_1 \cap E_{1\Sigma_o}$;
- l'ordre induit par la projection de b_2 : $\leq_2 \cap E_{2\Sigma_o}$;
- l'ordre induit par la composition : $\leq_D = \{(e, e') \in E_1^+ \times E_2^- \mid (\lambda(e), \lambda(e')) \in D\}$;
- l'ordre induit par les branchements : $\leq_{E^-_{1\boxdot 2} \rightsquigarrow (E_1^- \cup E_2^-)} \cup \leq_{(E_1^+ \cup E_2^+) \rightsquigarrow E^+_{1\boxdot 2}}$.

De même, $\leq_3$ peut se définir comme la fermeture transitive de l'union de $\leq_1$, de $\leq_2$, de $\leq_D$ et des branchements $\leq_{E_3^- \rightsquigarrow (E_1^- \cup E_2^-)}$ et $\leq_{(E_1^+ \cup E_2^+) \rightsquigarrow E_3^+}$. Nous cherchons, ensuite, à caractériser, la restriction de $\leq_3$ à E_3. Comme $\leq_1 \cap E^2_{1\Sigma_o} = \leq_1 \cap E^2_{3\Sigma_o}$ et $\leq_2 \cap E^2_{2\Sigma_o} = \leq_2 \cap E^2_{3\Sigma_o}$, nous obtenons que cette restriction donne exactement $\leq_{1\boxdot 2}$, ce qui conclut cette preuve. □

La figure 4.7 illustre cette proposition. Elle montre que la projection sur $\Sigma_o = \{a, b\}$ de la composition de $B(p_1)$ et $B(p_2)$, avec $D = \{(a,a), (c,b), (c,d)\}$ donne exactement la composition de la projection de $B(p_1)$ et de la projection de $B(p_2)$. En effet, $\widehat{\pi}_{\Sigma_o}(B(p_1) \boxdot_D B(p_2)) = b_4 = \widehat{\pi}_{\Sigma_o}(B(p_1)) \boxdot_D \widehat{\pi}_{\Sigma_o}(B(p_2))$.

Nous généralisons ensuite ce résultat aux expressions rationnelles. En effet, nous montrons, avec le théorème 4.8, qu'étant donné α, une expression rationnelle de $\mathbb{B}(\Sigma, D)$, il est très simple de trouver β, une expression rationnelle $\mathbb{B}(\Sigma, D)$ telle que le langage engendré par cette expression rationnelle soit exactement la projection du langage de α sur un alphabet Σ_o donné. En effet, il suffit simplement de choisir l'expression rationnelle $\beta = \widehat{\pi}_{\Sigma_o}(\alpha)$, qui est α dans laquelle chaque pomset générateur $p \in \Gamma(\alpha)$ est remplacé par $\widehat{\pi}_{\Sigma_o}(p)$.

Théorème 4.8 *Soient α, une expression rationnelle de $\mathbb{B}(\Sigma, D)$ et $\Sigma_o \subseteq \Sigma$. Alors :*

$$\widehat{\pi}_{\Sigma_o}(\mathcal{L}(\alpha)) = \mathcal{L}(\widehat{\pi}_{\Sigma_o}(\alpha))$$

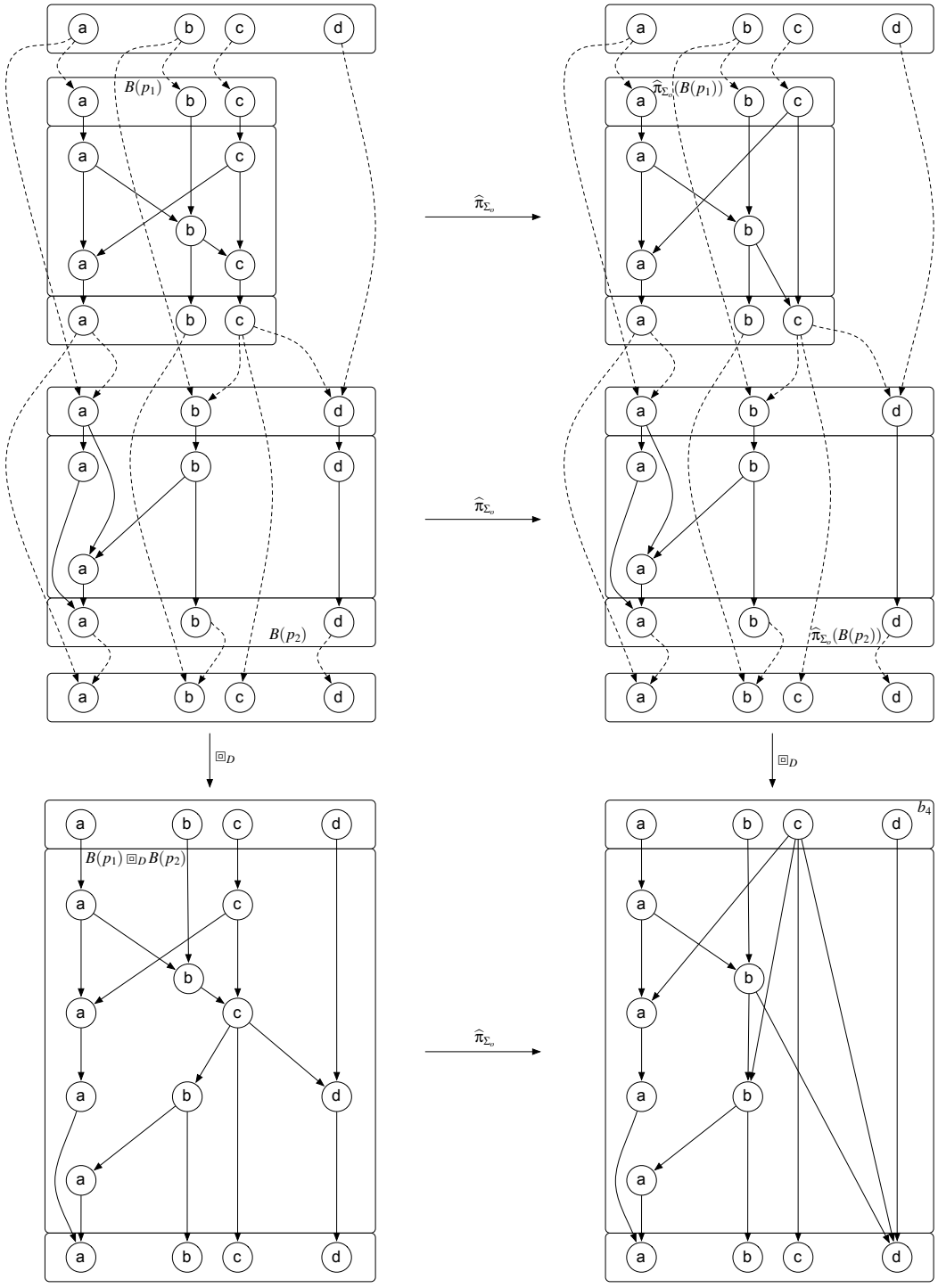


FIG. *4.7 – L'opérateur de projection est distributif sur celui de la composition pour les boxed pomsets, avec* $D = \{(a,a),(c,b),(c,d)\}$ *et* $\Sigma_o = \{a,b\}$.

Preuve: La proposition 4.7 indique que $\widehat{\pi}_{\Sigma_o}$ est un morphisme, ce qui donne le résultat directement. Cependant, afin de mieux comprendre comment utiliser les boxed pomsets, nous donnons une démonstration constructive de ce résultat, en s'appuyant sur la représentation sous la forme d'un automate $\mathcal{A}_\alpha$ structurellement équivalent, de l'expression rationnelle α.

D'une part, montrons que tous les mots de $\widehat{\pi}_{\Sigma_o}(\mathcal{L}(\alpha))$ sont dans $\mathcal{L}(\widehat{\pi}_{\Sigma_o}(\alpha))$. Pour cela, considérons un chemin ρ de $\mathcal{A}_\alpha$. Nous notons $\mathcal{L}(\rho) = b_1 \boxdot_D \ldots \boxdot_D b_n$, où $(b_i)_{1\leq i\leq k}$ est la séquence des boxed pomsets qui étiquettent ρ. Nous pouvons, alors,, appliquer la proposition 4.7 afin d'obtenir $\widehat{\pi}_{\Sigma_o}(\mathcal{L}(\rho)) = \widehat{\pi}_{\Sigma_o}(b_1) \boxdot_D \ldots \boxdot_D \widehat{\pi}_{\Sigma_o}(b_n)$. De plus, $\widehat{\pi}_{\Sigma_o}(b_1), \ldots, \widehat{\pi}_{\Sigma_o}(b_n)$ sont des boxed pomsets qui peuvent être trouvés le long d'un chemin de $\mathcal{A}_{\widehat{\pi}_{\Sigma_o}(\alpha)}$, l'automate structurellement équivalent à $\widehat{\pi}_{\Sigma_o}(\alpha)$. Ce qui veut dire, finalement, que $\widehat{\pi}_{\Sigma_o}(\mathcal{L}(\alpha)) \subseteq \mathcal{L}(\widehat{\pi}_{\Sigma_o}(\alpha))$.

D'autre part, montrons l'inclusion inverse. Pour cela, considérons un chemin ρ de $\mathcal{A}_{\widehat{\pi}_{\Sigma_o}(\alpha)}$. Nous avons alors $\mathcal{L}(\rho) = b_1 \boxdot_D \ldots \boxdot_D b_k$, avec $(b_i)_{1\leq i\leq k}$ la séquence de boxed pomsets rencontrés le long de ρ. Or, chaque b_i étiquette une transition de $\mathcal{A}_{\widehat{\pi}_{\Sigma_o}(\alpha)}$, ce qui signifie que $b_i = \widehat{\pi}_{\Sigma_o}(b'_i)$. Ainsi, nous pouvons appliquer la proposition 4.7 afin d'obtenir $\mathcal{L}(\rho) = \widehat{\pi}_{\Sigma_o}(b'_1 \boxdot_D \ldots \boxdot_D b'_k)$ De plus, les $(b'_i)_{1\leq i\leq k}$ forment une séquence présente le long d'un chemin de $\mathcal{A}_\alpha$. Ce qui veut dire, finalement, que $\mathcal{L}(\widehat{\pi}_{\Sigma_o}(\alpha)) \subseteq \widehat{\pi}_{\Sigma_o}(\mathcal{L}(\alpha))$. Ceci conclut la preuve de ce théorème. □

Considérons, par exemple, l'expression rationnelle $\alpha = B(p_1)B(a)^\star B(p_2)$. Le théorème 4.8 indique que la projection du langage de α est simplement le langage de l'expression rationnelle $\widehat{\pi}_{\Sigma_o}(\alpha) = \widehat{\pi}_{\Sigma_o}(B(p_1))(\widehat{\pi}_{\Sigma_o}(B(a)))^\star \widehat{\pi}_{\Sigma_o}(B(p_2))$. Les éléments de ce langage seront de la même forme que le boxed pomset b_4 de la figure 4.7, avec un nombre non borné d'événements étiquetés par a qui succèdent à ceux de $\widehat{\pi}_{\Sigma_o}(B(p_1))$ et précèdent ceux de $\widehat{\pi}_{\Sigma_o}(B(p_2))$. Remarquons alors, que la désencapsulation de ces boxed pomsets correspond exactement aux pomsets engendrés par la projection du langage de $p_1 a^\star p_2$, défini dans l'exemple de la partie 4.1.1. Nous montrons, dans la partie suivante, que ce résultat se généralise à la projection de toutes les expressions rationnelles de pomsets.

Expressions rationnelles de pomsets

Nous avons vu dans la partie précédente, d'une part, les propriétés liées aux expressions rationnelles de boxed pomsets. D'autre part, nous montrons, dans cette partie, les propriétés liées aux expressions rationnelles de pomsets.

Plus précisément, nous montrons qu'il existe un générateur rationnel fini qui engendre la projection de n'importe quel rationnel de pomset. Pour cela, dans la proposition 4.9, nous étendons aux langages les résultats montrés dans la proposition 4.6, sur le lien entre encapsulation et composition. Plus précisément, cette proposition montre que, pour toute expression rationnelle de $\mathbb{P}(\Sigma, D)$, il est équivalent d'encapsuler le langage de pomsets engendré ou de considérer le langage généré par l'encapsulation de chaque pomset qui l'engendre.

Proposition 4.9 *Soient α, une expression rationnelle de $\mathbb{P}(\Sigma, D)$ et $\Sigma_o \subseteq \Sigma$. Alors :*

$$\mathcal{L}(B(\alpha)) = B(\mathcal{L}(\alpha))$$

Preuve: Ce résultat est immédiat en utilisant le fait que B est un morphisme de monoïde, d'après la proposition 4.6. □

Finalement, nous donnons le résultat principal de cette partie. Ce résultat indique qu'il est possible de garder une représentation rationnelle de la projection d'un automate, modulo le choix d'un monoïde adapté : en l'occurence, celui des boxed pomsets. Plus précisément, le théoreme 4.10 indique que la projection du langage d'une expression rationnelle de $\mathbb{P}(\Sigma, D)$ est la désencapsulation de l'expression rationnelle obtenue en encapsulant, puis en projetant l'expression rationnelle initiale.

Théorème 4.10 *Soient α, une expression rationnelle de $\mathbb{P}(\Sigma, D)$ et $\Sigma_o \subseteq \Sigma$. Alors :*

$$\pi_{\Sigma_o}(\mathcal{L}(\alpha)) = U(\mathcal{L}(\widehat{\pi}_{\Sigma_o}(B(\alpha))))$$

Preuve: Nous utilisons les propositions montrées précédemment :

$$\begin{array}{rcll} \pi_{\Sigma_o}(\mathcal{L}(\alpha)) & = & \pi_{\Sigma_o}(U(B(\mathcal{L}(\alpha)))) & \text{(Prop. 4.4)} \\ & = & U(\widehat{\pi}_{\Sigma_o}(B(\mathcal{L}(\alpha)))) & \text{(Prop. 4.5)} \\ & = & U(\widehat{\pi}_{\Sigma_o}(\mathcal{L}(B(\alpha)))) & \text{(Prop. 4.9)} \\ & = & U(\mathcal{L}(\widehat{\pi}_{\Sigma_o}(B(\alpha)))) & \text{(Th. 4.8)} \end{array}$$

□

Ce théorème montre l'intérêt des boxed pomsets. En effet, pour toute expression rationnelle de pomsets de $\mathbb{P}(\Sigma, D)$ α, il n'existe pas, en général, d'expresssion rationnelle de $\mathbb{P}(\Sigma, D)$, dont la projection sur un alphabet d'actions observables forme le même langage. Cependant, il est trivial de construire une expression rationnelle de $\mathbb{B}(\Sigma, D)$, dont l'ensemble des boîtes internes des éléments engendrés forme le même langage : il suffit de choisir $\widehat{\pi}_{\Sigma_o}(B(\alpha))$. Il semble ainsi plus simple de concevoir des spécifications avec des pomsets et de manipuler des vues partielles de ces spécifications à l'aide des boxed pomsets.

Nous terminons cette partie en reprenant l'exemple de la partie 4.1.1, avec l'expression rationnelle $p_1 a^\star p_2$. Comme indiqué précédemment, la projection du langage de cette expression sur $\Sigma_o = \{a, b\}$ est exactement la désencapsulation du langage de l'expression rationnelle $\widehat{\pi}_{\Sigma_o}(B(\alpha)) = \widehat{\pi}_{\Sigma_o}(B(p_1))(\widehat{\pi}_{\Sigma_o}(B(a)))^\star \widehat{\pi}_{\Sigma_o}(B(p_2))$.

4.3 Projection et modèles finiment engendrés

Dans la partie précédente, nous avons montré qu'il existait un générateur rationnel fini qui permettait de représenter la projection d'un langage rationnel. Ce générateur est obtenu en encapsulant les pomsets générateurs de l'expression rationnelle, puis en les projetant. Nous montrons, maintenant, que ce générateur peut être utile pour décider quand la projection d'un rationnel reste rationnel. Plus précisément, nous caractérisons les expressions rationnelles de boxed pomsets dont le langage désencapsulé est un langage rationnel de pomsets.

Lemme 4.11 *Soient $b \in \mathbb{B}(\Sigma, D)$, un boxed pomset et $\{p_1, \ldots, p_k\}$, l'ensemble minimal de pomsets tels que $U(b^{|\Sigma|+1}) \in \langle\{p_1 \ldots p_k\}\rangle_{\mathbb{P}(\Sigma,D)}$. Alors, $U(b^{|\Sigma|+2}) \in \langle\{p_1 \ldots p_k\}\rangle_{\mathbb{P}(\Sigma,D)}$ si, et seulement si, $U(b^\star)$ est finiment engendré.*

Preuve: Notons b_i la i-ième instance du boxed pomset b dans le produit $b^{|\Sigma|+2}$, pour $i = \{1 \dots |\Sigma| + 2\}$. De même, étant donné un événement e de b, notons e_i l'événement correspondant dans b_i. Enfin, étant donné un événement e dans $b^{|\Sigma|+2}$, notons $P(e) \subseteq \Sigma$, l'ensemble des étiquettes qui apparaissent dans le passé causal de cet événement.

Clairement, pour tout e dans b, $P(e_{|\Sigma|}) = P(e_{|\Sigma|+1}) = P(e_{|\Sigma|+2})$. Or, la composition des boxed pomsets ne dépend que des étiquettes des différents événements impliqués. Ainsi, les mêmes connexions causales vont s'effectuer lors de la composition entre $b^{|\Sigma|}$ et b et lors de la composition entre $b^{|\Sigma|+1}$ et b. Ces mêmes connexions causales vont à nouveau s'effectuer lorsque l'on va considérer $b^\star$ et b.

Par conséquent, si la décomposition en pomsets premiers de la désencapsulation de $b^{|\Sigma|+1}$ contient les mêmes pomsets que celle de $b^{|\Sigma|+2}$, alors il en sera de même pour toute décomposition de $b^{|\Sigma|+i}$, pour $i \geq 2$. Par conséquent, la désencapsulation de $b^\star$ est finiment engendrée.

Inversement, si la décomposition en pomsets premiers de la désencapsulation de $b^{|\Sigma|+1}$ contient les mêmes pomsets que celle de $b^{|\Sigma|+2}$, alors il en sera de même pour toute décomposition de $b^{|\Sigma|+i}$, pour $i \geq 2$. Pour montrer cela, il faut considérer l'algorithme de décomposition en pomsets premiers donnée par la proposition 3.5. Clairement, dans ce cas, il existe une composante fortement connexe de $\mathcal{G}_{U(b^{|\Sigma|+i})}$ qui grossit strictement à chaque itération de i. Donc la désencapsulation de n'est pas finiment engendrée. □

Nous pouvons maintenant utiliser ce lemme pour montrer qu'il est décidable de savoir si la projection d'un rationnel de pomsets reste rationnel.

Théorème 4.12 *Savoir si la désencapsulation du langage d'une expression rationnelle de $\mathbb{B}(\Sigma, D)$ est le langage d'une expression rationnelle de $\mathbb{P}(\Sigma, D)$ est décidable.*

Preuve: Il suffit de calculer la décomposition en pomsets premiers de tous les éléments du langage de l'expression rationnelle de boxed pomsets dans laquelle on a remplacé les $\star$ par $(|\Sigma| + 1)$. Ensuite, on fait la même chose, mais en remplaçant les $\star$ par $(|\Sigma| + 2)$. Si les ensembles de pomsets premiers sont identiques, alors la désencapsulation de la projection du langage de l'expression rationnelle est finiment engendrée et il existe une expression rationnelle de pomsets qui a le même langage. □

Corrolaire 4.13 *Savoir si la projection d'un rationnel de $\mathbb{P}(\Sigma, D)$ reste un rationnel de $\mathbb{P}(\Sigma, D)$ est décidable.*

Preuve: Soient α, une expression rationnelle de pomsets et $\beta = \widehat{\pi}_{\Sigma_o}(B(\alpha))$, le générateur dont la désencapsulation du langage correspond exactement à la projection du langage de α sur Σ_o (d'après le théorème 4.10). Il suffit, alors, d'appliquer directement le théorème 4.12 sur l'expression rationnelle de boxed pomsets β. □

Nous pouvons maintenant utiliser le corollaire 4.13 pour faire de la vérification partielle de modèles d'ordre. Nous nous contentons du cas le plus simple du model-checking positif.

Proposition 4.14 *Soient (Σ, D), un alphabet concurrent, α, une expression rationnelle de $\mathbb{P}(\Sigma, D)$, $\Sigma_o \subseteq \Sigma$ et β, une expression corationnelle de $\mathbb{P}(\Sigma_o, D)$. Alors, le model-checking positif est décidable, c'est-à-dire que l'on sait décider si $\pi_{\Sigma_o}(\mathcal{L}(\alpha)) \subseteq \mathcal{L}(\beta)$.*

Preuve: Immédiat en appliquant le corollaire 4.13. Dans le cas où celui-ci donne une expression rationnelle de pomsets, il reste à appliquer la proposition 3.9 du chapitre 3. Dans le cas contraire, c'est-à-dire s'il n'existe pas d'expression rationnelle de pomsets dont le langage est exactement la projection du langage de pomsets du modèle, l'inclusion n'est pas valide puisque β est finiment engendré et donc, la réponse au test de model-checking est négative. □

La théorie de la vérification partielle est incomplète, puisqu'elle ne permet pas, pour le moment, de faire du model-checking négatif ou de considérer des formules d'ordres partiels. Cependant, dans le cas où la projection du modèle reste un rationnel de pomsets, il est tout à fait possible d'utiliser les techniques développées dans le chapitre 3 de ce document, consacrée à la vérification de modèles donnés sous forme d'expressions rationnelles de pomsets.

4.4 Conclusions et perspectives

Résumé

Dans ce chapitre, nous avons défini un nouveau modèle, appelé boxed pomset, qui étend celui des pomsets en modifiant les opérateurs de projection et de composition associés. Ce modèle possède de bonnes propriétés de projection, puisque la projection de rationnels de boxed pomsets reste des rationnels de boxed pomsets. De même, ce modèle reste très proche de celui des pomsets, puisque nous avons défini des opérateurs d'encapsulation et de désencapsulation qui permettent de passer, respectivement, des pomsets aux boxed pomsets et des boxed pomsets aux pomsets. Ces différentes propriétés font des boxed pomsets un outil théorique privilégié pour travailler avec des rationnels de pomsets et des projections. Cet outil nous a permis de donner deux résultats théoriques importants.

Le premier résultat théorique qui s'appuie sur la théorie des boxed pomsets est le suivant. Il indique que la projection du langage de n'importe quelle expression rationnelle de pomset peut se représenter, à une désencapsulation près, à l'aide d'un générateur rationnel fini, construit à partir d'une expression rationnelle de boxed pomsets. Ce résultat nous semble important, car il permet de manipuler très simplement les rationnels de pomsets et les projections, sans s'appuyer sur une borne quelconque du système sous-jacent. Ce générateur rationnel fini sera intensivement utilisé dans le chapitre 6 de ce document.

Le second résultat théorique qui s'appuie sur la théorie des boxed pomsets est le suivant. Il indique que, savoir si la projection d'un rationnel de pomsets reste un rationnel, est un problème décidable. Ce résultat, là encore, ne s'appuie sur aucune borne du système sous-jacent et généralise les résultats de [Genest 03] qui portent sur la projection de HMSC, en utilisant une technique de preuve complètement différente. Ce résultat est utile lorsque l'on cherche à faire du model-checking partiel positif, puisqu'il indique que ce problème de vérification est, lui aussi, décidable.

Perpectives

Les perspectives de ce chapitre s'organisent en deux parties. D'une part, une partie concerne les applications des techniques existantes pour les rationnels de pomsets (ou pour les HMSC) aux rationnels de boxed pomsets. D'autre part, une partie concerne la vérification partielle de modèles.

D'une part, une partie concerne le développement de la théorie des boxed pomsets en s'appuyant sur des techniques existantes pour les expressions rationnelles de pomsets ou les HMSC.

Tout d'abord, il est important de rappeler que le découpage en pomsets premiers constitue le coeur des techniques du chapitre précédent. A ce titre, ces résultats peuvent difficilement être reproductibles aux boxed pomsets, puisqu'il ne semble pas exister de décomposition minimale satisfaisante (à part celle qui isole chaque événement et qui n'est guère utile puisqu'elle ne permet pas de traduire nos problèmes vers les traces corationnelles). Dans ce cadre, faire de la vérification directement sur des modèles construits à partir d'expressions rationnelles de boxed pomsets semble difficile.

Ensuite, de nombreux travaux ont été réalisés sur les HMSC qui peuvent s'adapter, pratiquement directement, aux boxed pomsets. C'est, par exemple, le cas du problème de l'appartenance, étudié par Alur et Yannakakis dans [Alur 99], qui peut se traduire vers les boxed pomsets, en le problème de l'appartenance d'une observation partielle à la projection d'un modèle. Nous appelons ce problème "diagnostic" et nous l'étudions plus en détails dans le chapitre 6. De la même manière, les résultats concernant le pattern-matching de HMSC ([Genest 02a]) semblent se généraliser sans trop de difficulté au pattern-matching de projection de rationnels de pomsets.

Enfin, il serait intéressant de considérer le cas spécifique de la projection de HMSC causaux et de le généraliser au cas des CHMSC causaux. Ce cadre est différent, puisqu'il existe une notion de canaux de communication ayant une capacité et que l'on s'autorise à composer des morceaux de protocoles qui ne sont pas forcément clos par communication. Dans ce nouveau cadre, il serait intéressant de voir comment rajouter, aux boxed pomsets, une notion de canaux de communication, pour augmenter l'expressivité des modèles tout en gardant des propriétés d'analyses décidables.

D'autre part, une partie concerne la vérification partielle de modèles. En effet, dans ce chapitre, nous avons montré une méthode qui permet de faire uniquement du model-checking positif.

Tout d'abord, il serait intéressant de donner la complexité exacte lorsque l'on cherche à savoir si la projection d'un rationnel de pomsets reste un rationnel. Ceci nous permettrait de caractériser plus précisément la complexité du model-checking positif de vues partielles d'un modèle d'ordres partiels.

Enfin, la technique de model-checking positif que nous avons dévelopée est une extension directe des résultats montrés sur la décidabilité de la stabilité de la projection de rationnels de pomsets. Il nous semble qu'en construisant des structures un peu plus complexes, capables, par exemple, de générer uniquement une sous-partie finiment engendrée du langage de l'expression rationnelle du modèle, nous serions capables de résoudre le problème du model-checking pour une gamme plus large de modèles de boxed pomsets. Cette perspective nous semble

importante pour proposer des techniques efficaces de vérification de vues partielles de modèles d'ordres partiels.

Troisième partie

Supervision

Chapitre 5

Supervision avec observation complète

Nous avons vu, dans la partie I de ce document, différents modèles pour spécifier des systèmes parallèles et répartis et nous avons introduit un modèle mixte, localement asynchrone et globalement communicant qui les généralise. Ensuite, la partie II s'est attachée à décrire des techniques de vérification de tels modèles.

Ce chapitre constitue, avec le chapitre suivant, la partie III de cette thèse, consacrée à la *supervision*. Plus précisément, dans ce chapitre, nous nous intéressons aux analyses de supervision avec *observation complète*, lorsque les exécutions à analyser consistent en un très grand nombre d'événements. Ce qui signifie que nous allons étudier et développer des techniques qui ont vocation à s'exécuter en même temps (ou avec un léger délai) que le système à analyser s'exécute et qui doivent être *efficaces*.

Plus précisément, motivés par un traitement efficace de l'observation d'une exécution, dont la taille peut être volumineuse, nous nous intéressons, à deux problèmes.

Le premier problème consiste à compresser, lors de son observation, l'exécution d'un système parallèle et réparti. Cette compression, que l'on peut aussi voir comme un processus d'abstraction, va permettre de produire des macro-événements partiellement ordonnés.

Le second problème consiste à décider efficacement si cette observation correspond à un modèle donné. Pour cela, nous faisons un compromis de précision en décidant d'oublier l'ordre qui relie les événements observés ou produits par la technique précédente, en faisant simplement le décompte des différentes actions observées. A partir de cette approximation, nous caractérisons les modèles pour lesquels le test d'appartenance va pouvoir se faire de manière efficace.

Le chapitre suivant, lui aussi consacré à la supervision, se focalisera sur les analyses de supervision de systèmes que l'on ne peut observer que *partiellement*.

Contexte

Le problème de supervision a, en pratique, de nombreuses applications. En particulier, nous nous intéressons au cas de l'analyse de pannes dans les systèmes de télécommunication. Ces systèmes sont de plus en plus complexes, et font intervenir de nombreuses infrastructures matérielles et logicielles appartenant à plusieurs opérateurs. Dans ce contexte, trouver rapidement, lorsqu'une panne a lieu, d'où elle vient et qui en est responsable n'est pas seulement un problème technique important, mais aussi un enjeu économique majeur. En effet, de nos jours, un réseau téléphonique ou un fournisseur internet doit avoir une disponibilité continue, les usagers n'acceptant pas qu'une interruption de services puisse se prolonger, ne serait-ce que quelques heures.

Ainsi, afin de pouvoir réagir immédiatement en cas de panne, la plupart des infrastructures matérielles et logicielles des réseaux de télécommunication modernes sont capables d'enregistrer leurs activités courantes : les systèmes d'exploitation enregistrent tous les événements relatifs à la sécurité et aux accès bas niveau générés par les logiciels qu'ils exécutent, les serveurs reliés à internet enregistrent les différents événements provenant de leurs clients, les applications classiques enregistrent les erreurs d'exécution qu'elles rencontrent, les pare-feux et les passerelles réseau enregistrent le trafic suspect, les routeurs observent et analysent les paquets qu'ils redirigent ... En pratique, de nombreux protocoles existent, comme SNMP (pour "Simple Network Management Protocol" [Group 90]), permettant aux différentes infrastructures d'envoyer ce qu'elles enregistrent à un serveur qui va pouvoir centraliser toutes ces alarmes.

De plus, toutes ces entités ne se contentent pas d'enregistrer les événements qui sont liés à leurs propres exécutions. En effet, elles reçoivent et retransmettent des alarmes provenant de leurs voisins, qui, eux-aussi, font de même. Ce qui, au final, génère une propagation des différentes alarmes à travers tout le système considéré. Ainsi, une simple panne, localisée sur une seule entité logique ou physique peut générer une énorme quantité d'alarmes, provenant de tous les composants du système et qui sont enregistrées dans des fichiers appelés "logs" et dont la taille peut, par conséquent, devenir très grande. Dans ce cadre, lorsqu'une panne intervient, il devient évident qu'un être humain sera incapable d'analyser exhaustivement les logs produits.

Par conséquent, pour pouvoir étudier de tels logs et découvrir la ou les origines de la panne qui les a engendrés, il est nécessaire de développer des outils efficaces. Nous allons parler de ce genre d'outils dans ce chapitre. Plus précisément, nous allons décrire dans ce chapitre deux techniques qui permettent de simplifier l'interprétation de fichiers volumineux, directement issus de l'*observation exhaustive et complète* d'une exécution.

Organisation du chapitre

La première technique est décrite dans la partie 5.1. Elle consiste à abstraire, et donc, d'un certain point de vue, à compresser une observation d'un système parallèle et réparti. Cette abstraction est réalisée à la volée, c'est-à-dire au fur et à mesure que des événements sont enregistrés et ne s'appuie sur aucun modèle de système sous-jacent.

La seconde technique est décrite dans la partie 5.2. Elle consiste à approximer une observation, issue, par exemple, de la première technique décrite dans ce chapitre, par un décompte des différents macro-événements observés. Nous verrons ensuite qu'il est possible de caractériser des modèles qui permettent, à partir d'un décompte des actions d'une observation complète, de donner très rapidement un ensemble restreint d'explications possibles à cette observation.

Finalement, la partie 5.3 conclura cette partie en proposant quelques perspectives.

5.1 Abstraction

Le problème de l'abstraction consiste à agréger des événements de bas niveau (ou "concrets"), produits par les entités d'un système parallèle et réparti, pour former des événements de plus haut niveau (ou "abstraits"). Ce problème apparaît dans de nombreuses applications comme la supervision, le débogage, et plus généralement, lorsqu'il s'agit de confronter le comportement observable d'un programme réparti à un modèle dans lequel les événements de bas niveau sont connus et où le modèle se situe à un niveau supérieur d'abstraction.

Dans cette partie, nous nous plaçons dans le contexte de la supervision de systèmes parallèles et répartis. Nous nous donnons comme objectif que le résultat de l'abstraction *conserve les mêmes propriétés structurelles* que l'observation, c'est-à-dire reste un ordre partiel étiqueté. De plus, nous voulons que la technique d'abstraction puisse *être codée à la volée* par un observateur du système.

5.1.1 Etat de l'art

Il est bien connu, depuis les travaux fondateurs de Lamport ([Lamport 78]) que l'ensemble des événements d'une exécution d'un système parallèle et réparti s'organise sous la forme d'un ordre partiel de causalité, en considérant que deux événements sont ordonnés, s'ils ont lieu sur le même processus ou, s'ils sont séparés par un échange de messages ou par une synchronisation. Deux événements non causalement ordonnés sont dits indépendants ("concurrent" en anglais).

Une première proposition pour l'abstraction a été faite par Lamport dans [Lamport 86]. Elle propose qu'un événement abstrait précède un autre événement abstrait si, et seulement si, *tous* les événements concrets du premier événement abstrait précèdent *tous* les événements concrets du deuxième. Il est facile de montrer que l'ordre abstrait ainsi défini reste un ordre partiel. Le problème avec cette définition est le suivant : deux événements abstraits peuvent être indépendants alors qu'un événement concret du premier événement abstrait peut précéder causalement un événement concret du deuxième. Ce phénomène est contre-intuitif. Surtout, comme l'apparition d'une dépendance au niveau abstrait est exigeante, de nombreuses indépendances vont apparaître, détruisant la séquentialité des processus et rendant impossible la reconstruction de l'ordre concret à partir de l'ordre abstrait. La perte de cette séquentialité est problématique dans les applications de débogage, par exemple, puisqu'elle est complètement contre-intuitive.

Les travaux suivants, représentés, par exemple, par ceux de Kunz ([Kunz 93]), proposent une approche duale : un événement abstrait précède un autre événement abstrait si, et seule-

ment si, *un* événement concret du premier précède *un* événement concret du deuxième. Cette idée est naturelle d'un point de vue mathématique puisqu'il s'agit de la notion classique de quotient d'un graphe par une relation d'équivalence, les événements groupés formant les classes d'équivalence. Malheureusement, le quotient d'un ordre par une relation d'équivalence quelconque ne produit pas forcément un ordre (création de cycles par exemple). De notre point de vue, il est dommageable que la structure de la causalité dans les exécutions réparties change de nature suivant le niveau d'abstraction considéré. En effet, cela rend impossible tout processus d'abstraction hiérarchique.

La solution adoptée dans [Ahuja 90] consiste à imposer une restriction dans la façon de grouper les événements pour préserver la structure d'ordre partiel. Dans ces travaux, certaines classes d'équivalence sont considérées, celles définies comme les plus petits ensembles d'événements clos par communication (l'émission et la réception d'un message doivent être confinées dans la même classe) et sans trous (si deux événements produits par un processus donné sont dans une même classe, tous les événements intermédiaires produits sur ce processus le sont aussi). Il s'agit de la notion d'atome. Le prix à payer pour cette approche est que les atomes peuvent s'avérer très gros pour des exécutions complexes présentant de nombreux croisements imbriqués de messages. L'autre inconvénient, symétrique de l'approche de Lamport, est que deux événements indépendants l'un de l'autre peuvent se retrouver dans deux macro-événements ordonnés. Cet inconvénient semble mineur dans la mesure où la séquentialité des processus est préservée et où l'on peut reconstruire l'ordre concret initial.

Notre approche

Dans cette première partie de chapitre, nous choisissons de généraliser cette dernière approche. Nous commençons par formaliser la notion d'abstraction préservant l'ordre partiel, notion paramétrée par un typage des événements. Puis, guidés par l'application à la supervision, nous étudions la question nouvelle du calcul au vol de l'abstraction pendant l'exécution des processus. Les algorithmes présentés dans cette première partie de chapitre vont permettre d'utiliser le calcul d'une abstraction comme un filtre sur l'exécution et de traiter des exécutions de longueur arbitraire. Chaque événement produit sur un processus est envoyé par un message vers un abstracteur unique chargé de décider quand il a reçu suffisamment d'événements pour former un macro-événement. Il produit alors cet événement et la relation de causalité le reliant aux autres. Ces travaux ont été publiés dans [Gazagnaire 07a].

Organisation de la partie 5.1

Le reste de cette première partie est organisé ainsi : dans la partie 5.1.2, nous définissons la notion d'équivalence close par types et compatible avec l'ordre partiel (CTC-équivalence). La partie 5.1.3 présente le mécanisme d'extraction des événements et le codage de l'ordre par horloge vectorielle (filtrage). Enfin, l'algorithme d'abstraction, localisé sur un processus observateur et produisant les macro-événements au vol est décrit dans la partie 5.1.4.

5.1.2 Définition formelle de l'abstraction

Considérons un ensemble de n processus séquentiels, que l'on note $\{P_i\}_{1 \leq i \leq n}$, qui communiquent à l'aide d'un ensemble de canaux fiables unidirectionnels. Chaque processus produit

des événements atomiques, internes au processus (les événements observables). Nous les indexons par un couple d'entiers : le premier correspondant au numéro de processus sur lequel il a lieu, le second au nombre d'événements l'ayant précédé sur ce même processus. Nous notons ainsi $x_{i,j}$, le j-ième événement ayant lieu sur le processus P_i. L'ensemble des événements apparaissant sur un processus P_i dans une exécution sera noté X_i, et l'ensemble de tous les événements noté $X = \bigcup_{1 \leq i \leq n} X_i$.

Lamport a montré, dans [Lamport 78], que l'exécution X peut être structurée par une relation d'ordre (partiel) entre les événements, notée $\leq$, qui capture la causalité entre les différents événements : $x_{i,j} \leq x_{k,l}$ signifie que $x_{i,j}$ s'est produit forcément avant $x_{k,l}$.

Il a aussi été montré, dans [Mattern 89, Fidge 91], que cette relation peut être codée au vol pour les systèmes communicants, par une estampille vectorielle δ (un vecteur d'entiers), définie par $\delta(x) = (\mid I_X(x) \cap X_i \mid)_{i \in 1..n}$, avec $I_X(x)$ l'ensemble des prédécesseurs causaux de l'événement x : $I_X(x) = \{y \in X, y \leq x\}$.

$\delta(x)[i]$, la i-ième composante du vecteur $\delta(x)$, désigne le nombre d'événements qui précédent x sur P_i. Le plongement de l'ordre de causalité dans l'ordre sur les vecteurs d'entiers est attesté par la propriété suivante :

$$\forall x, y \in X, x \leq y \Leftrightarrow \forall i \in 1..n, \delta(x)[i] \leq \delta(y)[i]$$

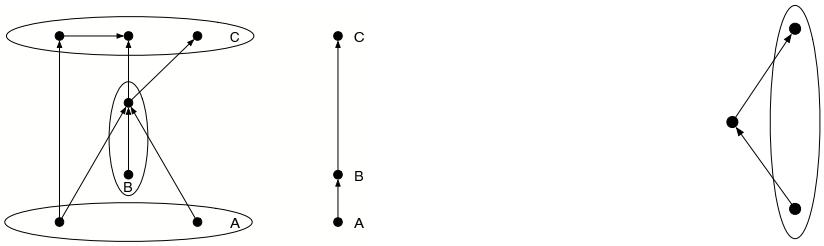
On notera par la suite, $max(\delta, \delta')$ le vecteur δ_{max} tel que $\delta_{max}[i] = max(\delta[i], \delta'[i])$ pour tout $i \leq n$. De plus, $\delta < \delta'$ si, et seulement si, $\delta[i] < \delta'[i]$ pour tout $i \leq n$. On rappelle la définition de la couverture : $\forall x, y \in X, x \prec y$ si, et seulement si, $x \neq y \wedge \exists z \in X, x \leq z \leq y \Rightarrow (z = x \vee z = y)$. On dira que $\prec$ est la couverture de $\leq$.

Abstraire une structure consiste à fusionner des événements entre eux. Nous modélisons ceci par une relation d'équivalence $\sim \subseteq X \times X$. Deux événements seront fusionnés si ils sont équivalents. Ensuite, de manière classique, étant donné l'équivalence $\sim$, nous notons $[x]_\sim$ la classe d'équivalence de x. La classe d'équivalence de x représente l'ensemble des événements qui seront fusionnés avec x lors de l'abstraction. Le résultat de l'abstraction est le quotient de X par $\sim$, noté $X/_\sim$, c'est-à-dire l'ensemble des classes d'équivalence. Un exemple est donné à la Figure 5.1(a). Par la suite, nous identifierons une classe de $X/_\sim$ avec l'ensemble des événements qui la compose.

Définition 5.1 *Soit $(X, \prec)$, la couverture de la causalité d'une exécution répartie. On note $\prec_\sim \subseteq X/_\sim \times X/_\sim$ la plus petite relation qui vérifie $x \prec y \implies [x]_\sim \prec_\sim [y]_\sim$. De plus, on notera $\leq_\sim$ la fermeture transitive de $\prec_\sim$.*

Malheureusement, le passage au quotient ne préserve pas la structure de l'ordre sous-jacent. En effet, comme montré sur la Figure 5.1(b), le quotient d'un ordre n'est pas toujours antisymétrique, ni transitif. Notre objectif est de conserver une structure d'ordre sur l'objet abstrait. Pour cela, nous devons restreindre la relation $\sim$ pour qu'elle soit compatible avec le quotient.

Définition 5.2 *Une relation d'équivalence $\sim \subseteq X \times X$ est* compatible *avec la couverture de l'exécution $(X, \leq)$ (on dira plus simplement qu'elle est compatible avec l'exécution $(X, \leq)$) si*



(a) Exemple d'abstraction (b) cycle

FIG. *5.1* – Figure a (gauche) : les événements entourés forment les macro-événements. Figure b (droite) : un quotient qui ne préserve par l'anti-symétrie.

elle satisfait la condition suivante : pour tout $x_0 \dots x_{2n}$ *tels que* $x_{2i} \sim x_{2i+1}$ *et* $x_{2i+1} \prec x_{2(i+1)}$ *pour* $0 \leq i < n$*, si* $x_0 = x_{2n}$*, alors, on a* $x_1 \sim \dots \sim x_{2n}$*.*

On peut remarquer, de manière immédiate, que les deux abstractions triviales, correspondant à $\forall x \in X$, $[x]_\sim = \{x\}$ et $\forall x \in X$, $[x]_\sim = X$, sont compatibles avec n'importe quel ordre partiel. Intuitivement, le quotient d'une exécution par une relation d'équivalence compatible ne crée pas de cycles.

Proposition 5.3 *Soit* $\sim$*, une relation d'équivalence compatible avec* $(X, \leq)$*. Alors,* $\leq_\sim$ *est une relation d'ordre sur* $X/_\sim$*.*

Preuve: Tout d'abord, la réflexivité de $\leq_\sim$ est héritée de la réflexivité de $\leq$.

Ensuite, supposons qu'il existe deux classes A et B telles que $A \leq_\sim B$ et $B \leq_\sim A$. Par définition de $A \leq_\sim B$, on a : $\exists x_1 \dots x_{2n}$ tels que $x_{2i} \sim x_{2i+1}$ pour $0 < i < n$ et $x_{2i+1} \prec x_{2(i+1)}$ pour $1 \leq i < n$, avec $x_1 \in A$ et $x_{2n} \in B$. De même, par définition de $B \leq_\sim A$, on a : $\exists x'_1 \dots x'_{2m}$ tels que $x'_{2i} \sim x'_{2i+1}$ pour $0 < i < n$ et $x'_{2i+1} \prec x'_{2(i+1)}$ pour $1 \leq i <$, avec $x'_1 \in B$ et $x'_{2m} \in A$. Comme $x_1 \sim x'_{2m}$ et $x_{2n} \sim x'_1$, on peut appliquer le fait que $\sim$ est compatible. On a donc $x_1 \sim \dots \sim x_{2n} \sim x'_1 \sim \dots \sim x'_{2m}$ et donc $A = B$.

Enfin, $\leq_\sim$ est définie comme étant la fermeture transitive de $\prec_\sim$. Elle est donc transitive par construction. □

La notion de compatibilité développée ici implique la notion d'abstraction convexe, développée par Basten et al. dans [Basten 97]. Cependant, la notion d'abstraction convexe est trop fine pour que la structure d'ordre partiel soit préservée. En conséquence, ces travaux ne permettent pas de construire une abstraction qui soit un ordre partiel. Nous cherchons maintenant une famille d'abstractions compatibles non triviales. Nous proposons pour cela d'étiqueter les événements.

Définition 5.4 *Considérons un ensemble* T *de types et une fonction de typage* $\mathcal{T}$ *étiquetant les événements de* X*. La relation d'équivalence* $\sim$ *est dite* close par types *si,* $\forall x, y \in X, \mathcal{T}(x) = \mathcal{T}(y) \implies x \sim y$*. Une relation d'équivalence compatible avec* $(X, \leq)$ *et close par types sera appelée une* CTC-équivalence *pour* $(X, \leq)$*.*

Un premier exemple de typage est le suivant : on associe à chaque événement, son numéro de processus : $\mathcal{T}(x_{i,j}) = i$. L'équivalence close par types correspondante construira le graphe de communication de l'exécution, c'est-à-dire le graphe orienté dont les sommets sont les processus, et pour lequel il existe un arc entre deux processus s'ils interagissent par échange de messages ou par synchronisation utilisant de la mémoire partagée. Un graphe de communication peut être cyclique, une relation close par types n'est donc pas, a priori, compatible.

Un autre exemple de typage est de considérer une fonction qui donne un identifiant unique aux messages échangés. L'équivalence close par types correspondante consistera à fusionner les paires émission-réception d'un même message et on obtiendra un graphe orienté dont les sommets correspondent aux messages et sont reliés si les envois de messages correspondant sont causalement reliés.

Proposition 5.5 *L'ensemble des CTC-équivalences pour l'exécution $(X, \leq)$ est ordonné par inclusion en disant qu'une équivalence est plus fine qu'une autre si, chaque classe d'équivalence de la première est incluse dans une classe d'équivalence de la seconde. L'ensemble des CTC-équivalences pour $(X, \leq)$, ordonné par la relation d'ordre définie ci-dessus, forme un treillis.*

Preuve: L'ensemble des équivalences de $X \times X$ forme un treillis (appelé le treillis des partitions de X). Il suffit donc de montrer que l'ensemble des CTC-équivalences admet un unique élément minimal et maximal. L'élément maximal est la relation universelle $\forall x \in X,\ [x]_{\sim_{max}} = X$, qui est clairement CTC. Il reste à montrer l'existence d'un unique élément minimal $\sim_{min}$. Supposons qu'il existe deux éléments minimaux, $\sim_1$ et $\sim_2$. Ces équivalences sont incomparables, il existe, donc, deux classes A et B telles que $A \cap B \neq \emptyset$ (avec $A \in X/_{\sim_1}$ et $B \in X/_{\sim_2}$). Choisissons x_1 dans $A/(A \cap B)$, x_2 dans $B/(A \cap B)$ et x_3 dans $A \cap B$. $x_1 \leq_{\sim_1} x_3$ et $\sim_1$ minimal impliquent qu'il existe $y_1 \dots y_{2n}$ tels que $\mathcal{T}(y_{2i}) = \mathcal{T}(y_{2i+1})$ pour $0 < i < n$ et $y_{2i+1} \prec y_{2(i+1)}$ pour $1 \leq i < n$, avec $\mathcal{T}(y_1) = \mathcal{T}(x_1)$ et $\mathcal{T}(y_{2n}) = \mathcal{T}(x_3)$. Ceci peut être répété pour $x_3 \leq_{\sim_2} x_2$, $x_2 \leq_{\sim_2} x_3$ et $x_3 \leq_{\sim_1} x_1$. Les "chaînes" construites ne dépendant ni de $\sim_1$ ni de $\sim_2$, on peut donc appliquer la propriété de compatibilité qui nous dit que $x_1 \sim_1 x_3 \sim_1 x_2$ (ou de manière équivalente $x_1 \sim_2 x_3 \sim_2 x_2$) et donc $A = B$ et $\sim_1 = \sim_2 = \sim_{min}$. □

L'élément minimal est appelé la *décomposition atomique* de l'exécution répartie $(X, \leq)$. Dans le cas particulier où il y a uniquement des échanges de messages (et pas de synchronisations), on peut associer une fonction de typage qui associe un même type à l'envoi et à la réception d'un même message. Les classes de la CTC-équivalence $\sim_M$ correspondante, formant une partition de X, sont identiques à celles formées en utilisant la notion d'atome de Ahuja et al. ([Ahuja 90]). De plus, Hélouët et Le Maigat ont montré dans [Hélouët 00b] que l'on pouvait calculer la décomposition en atomes d'une exécution $(X, \leq)$ en temps $O(|X|^2)$, où $|X|$ est le nombre d'événements contenus dans X. Dans la partie suivante, nous généralisons cette approche à toute CTC-équivalence pour $(X, \leq)$, et nous proposons un algorithme centralisé qui permet d'abstraire une exécution à la volée, ce qui permet de traiter effectivement des exécutions de taille arbitraire.

5.1.3 Abstraction et horloges vectorielles

Dans cette partie, nous donnons une solution algorithmique au processus d'abstraction défini de manière formelle dans la partie précédente. Chaque processus est équipé d'un *cap-*

teur qui implante le mécanisme d'horloges vectorielles en interceptant les messages émis et reçus et les synchronisations. Lorsqu'un événement a lieu sur un processus, le capteur associé envoie un message à un *observateur* global. Le message est étiqueté par l'horloge vectorielle et le type de l'événement observé. Enfin, l'observateur global est chargé de produire, à la volée (c'est-à-dire en même temps qu'il reçoit les messages provenant des différents capteurs), l'abstraction de l'exécution qui est en train de se dérouler sur les différents processus observés.

L'algorithme 5.1 définit le comportement du capteur attaché au processus P_i. Il reprend le calcul classique des horloges vectorielles en lui ajoutant une communication avec l'observateur global, quand cela est nécessaire (ligne 4) et un traitement des événements de synchronisation (lignes 14-18). Pour cela, nous ajoutons aux variables partagées, une horloge vectorielle qui contiendra l'horloge du dernier processus qui y a accédé (en lecture ou en écriture).

Algorithme 5.1 <**Capteur**> Gestion des événements liés au processus P_i

Entrées: n le nombre de processus, i le numéro du processus observé, $\mathcal{T}$ une fonction qui associe à chaque événement observable un type.

1: $\delta \leftarrow 0^n$;
2: **boucler**
3: **si** un événement e observable se produit sur P_i **alors**
4: Envoyer le message $(\mathcal{T}(e), \delta)$ à l'observateur global ;
5: $\delta[i] \leftarrow \delta[i] + 1$;
6: **fin si**
7: **si** un message (m, δ_m) provenant du capteur associé à P_j est reçu **alors**
8: Transmettre le message m à P_i ;
9: $\delta \leftarrow \max(\ \delta,\ \delta_m\ )$;
10: **fin si**
11: **si** P_i veut envoyer un message m à P_j **alors**
12: Envoyer le message (m, δ) au capteur associé à P_j ;
13: **fin si**
14: **si** P_i veut accéder à une variable partagée x **alors**
15: Récupérer l'horloge δ_x associée à x ;
16: $\delta \leftarrow \max(\ \delta,\ \delta_x\ )$; $\delta_x \leftarrow \delta$;
17: Accéder à la variable partagée ;
18: **fin si**
19: **fin boucle**

L'observateur global reçoit donc, des capteurs locaux, les différents événements observés, sans perte mais aussi sans garantie d'ordre : il peut y avoir des "trous" dans l'observation car certains messages ne sont pas encore arrivés du fait de l'asynchronisme des communications. Pour gérer cette contrainte, nous reprenons la notion d'horloges vectorielles pour des macro-événements introduite dans [Basten 97]. Ainsi, étant donné un ensemble A non vide d'événements concrets que l'on regroupe en un unique macro-événement, nous lui associons deux horloges vectorielles $\Delta^-(A)$ et $\Delta^+(A)$, définies localement, c'est-à-dire qu'elles ne dépendent que des horloges vectorielles des événements contenus dans A.

Définition 5.6 $\Delta^-(A)[i] = min(\{+\infty\} \cup \{\delta(x_{i,j})[j] - 1 \mid x_{i,j} \in A\})$ *est le plus grand numéro d'apparition d'un événement précédant A sur l'instance* P_i *(et* $+\infty$ *s'il n'y a pas d'événement ayant lieu sur* P_i*).* $\Delta^+(A)[i] = max\{\delta(x)[i] \mid x \in A\}$ *est le plus grand numéro d'apparition d'un événement sur l'instance* P_i *vu par un événement de A.* $loc(A) = \{i \mid x_{i,j} \in A\}$ *est l'ensemble des processus impliqués dans l'exécution du macro-événement A.*

Nous nous intéressons, maintenant, à la relation de *précédence faible*, notée $\rightarrow$ définie de la manière suivante : $A \rightarrow B$ si, et seulement si, il existe $e \in A$, $e' \in B$ tels que $e \leq e'$. [Basten 97] a montré que cette relation possède de bonnes propriétés algorithmiques car on peut coder $\rightarrow$ ainsi que *loc* à l'aide des deux horloges Δ^- et Δ^+. En effet, $loc(A)$ peut être réécrit en l'ensemble $\{i \in \{1..n\} \mid \Delta^-(A)[i] < \Delta^+(A)[i]\}$ et $A \rightarrow B$ est équivalent à ce qu'il existe $i \in loc(A)$ tel que $\Delta^-(A)[i] < \Delta^+(B)[i]$. Nous noterons $\rightarrow^*$ la fermeture transitive de la relation $\rightarrow$. Soit $\mathcal{T}$, une fonction de typage sur les macro-événements définie comme suit : pour tout $A \subseteq X$, $\mathcal{T}(A)$ est l'ensemble $\{\mathcal{T}(x) \mid x \in A\}$. Nous notons $A \rightarrow_{\mathcal{T}} B$ si, et seulement si, $A \rightarrow B$ ou $\mathcal{T}(A) \cap \mathcal{T}(B) \neq \emptyset$. Enfin, nous noterons $\rightarrow^*_{\mathcal{T}}$, la fermeture transitive de $\rightarrow_{\mathcal{T}}$.

Etant donné une CTC-équivalence $\sim$ pour une exécution $(X, \leq)$, la proposition suivante montre que l'on peut utiliser les bonnes propriétés algorithmiques de $\rightarrow$ (que l'on peut coder en une comparaison de vecteurs d'horloge) pour calculer $\leq_\sim$.

Proposition 5.7 *Soit* $\sim$*, la CTC-équivalence minimale pour* $(X, \leq)$*. Alors,* $\rightarrow^*_{\mathcal{T}} = \leq_\sim$.

Preuve: On montre ce résultat par induction sur X. Il suffit donc de montrer que, pour toute observation partielle $Y \subseteq X$, on a : si il existe $A, B, C \in Y/_\sim$ et $x \in X - Y$ tels que $A \rightarrow^* B \rightarrow_{\mathcal{T}} \{x\} \rightarrow_{\mathcal{T}} C \rightarrow^* A$ alors A, B, C et x sont dans un même macro-événement $D \in X/_\sim$.

Supposons qu'il existe $A \rightarrow^* B \rightarrow_{\mathcal{T}} \{x\} \rightarrow_{\mathcal{T}} C \rightarrow^* A$.

Alors, il existe $A, B_1, \ldots, B_k, C, C_1, \ldots, C_l, x$ tels que :

$$A \rightarrow B_1 \rightarrow \ldots \rightarrow B_k \rightarrow_{\mathcal{T}} \{x\} \rightarrow_{\mathcal{T}} C_1 \rightarrow \ldots \rightarrow C_l \rightarrow A$$

Il existe donc $x_1, \ldots, x_N \in X$ (pas forcément distincts) tels que $x_{2i} \prec x_{2i+1}$ et $\mathcal{T}(x_{2i+1}) = \mathcal{T}(x_{2(i+1)})$, avec au moins un x_i dans chaque $B, A_1, \ldots, A_k, \{x\}, C_1, \ldots, C_l$. On applique alors la propriété de compatibilité qui nous donne le résultat voulu. □

Nous donnons ensuite un résultat important concernant l'abstraction d'exécutions partiellement observées.

Définition 5.8 *Soit* $\sim$*, une CTC-équivalence compatible avec une exécution* $(X, \leq)$*. Pour toute observation partielle* $Y \subseteq X$*, et pour tout macro-événement A de* $Y/_\sim$*, A est dit* stable *pour l'observation Y si, de manière inductive :*

(i) A est clos par types, c.a.d $\{x \in X \mid \exists x' \in A,\ \mathcal{T}(x) = \mathcal{T}(x')\} \subseteq A$*,*

(ii) A est convexe, c.a.d. $\{x \in X \mid \exists y, y' \in A,\ y \leq x \leq y'\} \subseteq A$*,*

(iii) tous les événements du passé causal de A sont dans des macro-événements qui sont stables.

On note $Stable(Y/_\sim)$ *l'ensemble de tels A et* $Unstable(Y/_\sim)$ *son complément dans* $Y/_\sim$.

Proposition 5.9 *Soient $\sim$, une CTC-équivalence compatible avec une exécution $(X, \leq)$, $Y \subseteq X$, une observation partielle et A, un macro-événement de $Y/_\sim$. Alors, A est stable pour Y si, et seulement si, A est un macro-événement de $X/_\sim$.*

Preuve: Soit $Y \subseteq X$, une observation quelconque. Nous allons montrer la proposition précédente par induction sur la structure de $(Y/_\sim, \leq_\sim)$.

Prenons un élément minimal A de cette structure. Supposons que A ne soit pas un macro-événement de $X/_\sim$: il existe, donc, un macro-événement B de $X/_\sim$ tel que $A \subseteq B$. A vérifie (i) et (ii), il est donc convexe et clos par types. Il existe donc un $x_A \in A$ et un $x_B \in B$ tels que $x_B \leq x_A$. Ce qui est impossible car A est minimal. A est donc stable.

Prenons ensuite un macro-événement A non-minimal vérifiant (i) et (ii). On applique l'hypothèse d'induction qui nous dit que tous ses prédécesseurs sont stables. De plus, A est convexe et clos par types. De la même manière, on en déduit que A est stable. □

La partie stable d'une exécution le restera, quoi qu'il se passe par la suite. Ainsi, dès qu'un macro-événement est détecté comme étant stable, il peut être produit par l'observateur. De plus, comme l'abstraction engendrée reste un ordre partiel, on peut coder ce macro-événement par une horloge vectorielle classique, en comptant cette fois-ci le nombre de macro-événements déjà générés sur chaque instance. On peut alors voir un macro-événement A comme un événement normal ayant lieu simultanément sur les instances de $loc(A)$, un peu à la manière des événements locaux d'une synchronisation. Dans la partie suivante, nous définissons un observateur qui produit à la volée des macro-événements et un mécanisme qui permet de les ordonner.

5.1.4 Algorithmes pour l'observateur

Les événements observés par les capteurs sont reçus par l'observateur de manière complètement asynchrone, sans garantie de respect de l'ordre d'émission. Pour gérer cette difficulté, nous modélisons l'exécution globale par un ensemble X non borné et les événements observés par une famille d'ensembles d'événements $\emptyset = Y_0 \subset Y_1 \subset \ldots \subset Y_{|X|} = X$. Y_i correspond à l'ensemble des événements observés lorsque l'observateur global a reçu i messages provenant des capteurs. Les événements sont typés, et on cherche à construire à la volée $X/_\sim$, avec $\sim$ la CTC-équivalence minimale pour $(X, \leq)$.

L'algorithme 5.2 est la boucle principale de l'observateur. Il est appelé chaque fois qu'un nouveau message est reçu. Lorsque le k-ième message est reçu, l'observateur encapsule l'événement associé dans un macro-événement singleton A (ligne 12), puis il quotiente l'union de $Unstable(Y_{k-1}/_\sim)$ et de A (ligne 13). Finalement, il extrait les éléments stables de $\big(Unstable(Y_{k-1}/_\sim) \cup \{A\}\big)/_\sim$ afin de les fournir à l'utilisateur et il construit $Unstable(Y_k/_\sim)$ (ligne 14). Pour des raisons techniques, un macro-événement A sera codé comme la donnée de deux horloges $\Delta^-(A)$ et $\Delta^+(A)$, d'un entier $Size$ qui contiendra le nombre d'événements concrets agrégés dans A, et d'un multi-ensemble $\mathcal{T} = \{(t_i, n_i)\}_{i \leq \beta}$ où n_i est le nombre d'événements dans A ayant le type t_i et $\beta \leq k$ le nombre de types associés aux événements de $Unstable(Y_k/_\sim)$.

L'algorithme 5.3 insère le macro-événement singleton A dans $Unstable(Y_{k-1}/_\sim)$ et en calcule le quotient par $\sim$. Rappelons que n est le nombre de processus observés et que β est le

Algorithme 5.2 <**Observateur**> Programme principal

Entrées: i le numéro de processus sur lequel a lieu l'événement reçu, t son type et δ l'horloge vectorielle qui lui est associée.

Sorties: A est un macro-événement contenant l'unique événement considéré.

1: SINGLETON(i, t, δ) :=
2: $A.\Delta^+ \leftarrow \delta$;
3: $A.\Delta^- \leftarrow (+\infty)^n$;
4: $A.\Delta^-[i] \leftarrow A.\Delta^+[i] - 1$;
5: $A.Size \leftarrow 1$;
6: $A.\mathcal{T} \leftarrow (t, 1)$;
7: **renvoyer** A ;

Entrées: n le nombre de processus, $\mathcal{B}$ une fonction qui à chaque type associe le nombre d'événements étiquetés par ce type.

8: MAIN(n, $\mathcal{B}$) :=
9: $Memory \leftarrow \emptyset$; $MaxStableEvent \leftarrow 0^n$;
10: **boucler**
11: **si** le message (t, δ) numéro k provenant du capteur associé à P_i est reçu **alors**
12: $A \leftarrow$ SINGLETON(i, t, δ) ;
/* $Memory = Unstable(Y_{k-1}/_\sim)$ * /
13: $Memory \leftarrow$ INSERT(A, $Memory$) ;
/* $Memory = \big(Unstable(Y_{k-1}/_\sim) \cup \{A\}\big)/_\sim$*/
14: $(Memory, MaxStableEvent) \leftarrow$ EXTRACT($Memory$, $MaxStableEvent$, $\mathcal{B}$) ;
/* $Memory = Unstable(Y_k/_\sim)$ */
15: **fin si**
16: **fin boucle**

nombre de types différents apparaissant dans $Unstable(Y_{k-1}/_\sim)$. On a, alors, les résultats de complexité suivants : la fusion de macro-événements (ligne 1 à 5) se fait en temps $O(n+\beta)$, les tests $A \rightarrow_\mathcal{T} B$ (lignes 9 et 10) se font en $O(n+\beta)$ grâce au codage de la relation de précédence faible à l'aide des horloges Δ^- et Δ^+. Le choix d'une linéarisation de $\big(Unstable(Y_{k-1}/_\sim), \leq_\sim\big)$ (ligne 7) est un peu complexe car il faut reconstruire la fermeture transitive de $\rightarrow_\mathcal{T}$ (qui est égale à $\leq_\sim$ par la Proposition 5.7), ce qui peut être fait en $O(n\alpha^2)$ en utilisant un algorithme de décomposition en niveaux, où α est la taille maximale de $Unstable(Y_{i-1}/_\sim)$, pour $1 \leq i \leq k$ (et donc $\alpha \leq k$). Le calcul des horloges vectorielles (ligne 8) est classique : il consiste à calculer, pour chaque élément de L, le maximum de ces prédécesseurs immédiats par $\rightarrow_\mathcal{T}$, ce qui est fait en $O(n\alpha^2)$. Au total, l'algorithme INSERT a une complexité en $O(n\alpha^2 + \beta\alpha)$.

L'algorithme 5.4 extrait de $\big(Unstable(Y_{k-1}/_\sim) \cup \{A\}\big)/_\sim$ les macro-événements stables et construit $Unstable(Y_k/_\sim)$. Les tests, effectués ligne 6, correspondent aux points (i), (ii) et (iii) de la Proposition 5.9. La condition (i) nous impose de rajouter une contrainte supplémentaire sur le type d'abstraction considéré. En effet, pour savoir si un macro-événement est clos par types, il faut avoir des informations sur le type de tous les événements qui pourraient arriver, ce qui est impossible dans le cas général. Nous supposons donc, ici, que l'on connaît une fonction $\mathcal{B}$ qui associe à chaque type, le nombre d'événements ayant ce type. Cette supposition

Algorithme 5.3 <**Observateur**> INSERT(A, $Memory$)

Entrées: A et B sont deux macro-événements.
Sorties: A est modifié et contient l'union de A et B.

FUSION(A, B) :=
  $A.\Delta^- \leftarrow \min(\ A.\Delta^-,\ B.\Delta^-\ )$;
  $A.\Delta^+ \leftarrow \max(\ A.\Delta^+,\ B.\Delta^+\ )$;
  $A.Size \leftarrow A.Size + B.Size$;
  $A.\mathcal{T} \leftarrow A.\mathcal{T} \cup B.\mathcal{T}$;

Entrées: A est le nouveau singleton à ajouter à $Memory = Unstable(Y_{k-1}/_\sim)$.
Sorties: $\big(Unstable(Y_{k-1}/_\sim) \cup \{A\}\big)/_\sim$.

INSERT(A, $Memory$) :=
  $L \leftarrow$ Choisir une linéarisation de $(Memory, \rightarrow_{\mathcal{T}}^*)$;
  Parcourir L pour calculer une horloge vectorielle $\delta(B)$ pour chaque $B \in L$;
  $ImmPred \leftarrow \{B \mid A \rightarrow_{\mathcal{T}} B\}$;
  $ImmSucc \leftarrow \{B \mid B \rightarrow_{\mathcal{T}} A\}$;
  $Pred \leftarrow \{B \mid \exists C \in ImmPred,\ \delta(B) \leq \delta(C)\}$;
  $Succ \leftarrow \{B \mid \exists C \in ImmSucc,\ \delta(C) \leq \delta(B)\}$;
  pour tout $B \in Pred \cap Succ$ **faire** FUSION(A,B) **fin faire** ;
  renvoyer $\big(M - (Pred \cap Succ)\big) \cup \{A\}$;

Algorithme 5.4 <**Observateur**> EXTRACT($Memory$, $MaxStableEvent$, $\mathcal{B}$)

CLOSE(B, $\mathcal{B}$) := **pour tout** $t^i \in B.\mathcal{T}$ **faire** $\mathcal{B}(t) = i$; **fin faire**
CONVEX(B) := $\sum_{i \in loc(B)} (B.\Delta^+[i] - B.\Delta^-[i]) = Size(B)$;
MINIMAL(B, $MaxStablevent$) :=
  pour tout $i \in loc(B)$ **faire** $B.\Delta^-[i] = MaxStableEvent[i]$; **fin faire**

Entrées: Un ensemble de macro-événements $Memory = \big(Unstable(Y_{k-1}/_\sim) \cup \{A\}\big)/_\sim$, un vecteur d'entiers $MaxStableEvent$ qui contient, pour chaque processus, le numéro de l'événement maximal ayant déjà été extrait après avoir reçu $k-1$ messages, une fonction $\mathcal{B}$ qui associe, à chaque type, son nombre d'occurrences.
Sorties: L'ensemble $Unstable(Y_k/_\sim)$ et un vecteur d'entiers $MaxStableEvent$ qui contient, pour chaque processus, le numéro de l'événement maximal qui a déjà été extrait après avoir reçu k messages.

EXTRACT($Memory$, $MaxStableEvent$, $\mathcal{B}$) :=
  tant que il existe B dans $Memory$ tel que
      CLOSE(B, $\mathcal{B}$) & CONVEX(B) & MINIMAL(B, $MaxStableEvent$) **faire**
    pour tout $i \in loc(B)$ **faire** $MaxStableEvent[i] \leftarrow B.\Delta^+[i] + 1$ **fin faire** ;
    $Memory \leftarrow Memory - \{B\}$;
    Produire le macro-événement stable B ;
  fin faire
  renvoyer ($Memory$, $MaxStableEvent$) ;

n'est pas si restrictive, elle permet, par exemple, d'exprimer la notion d'atome définie dans [Ahuja 90], avec $\mathcal{B}(t) = 2$ si t est associé à l'envoi et à la réception d'un même message et avec $\mathcal{B}(t) = 1$, sinon. La complexité de CLOSE est en $O(\beta)$, celle de la ligne 9 en $O(n^2)$. La complexité de EXTRACT est donc en $O(\alpha n^2 + \alpha\beta)$.

Proposition 5.10 *Soient n, le nombre de processus observés, $\alpha = |Unstable(Y/_k/_\sim)|$ et β, le nombre de types associés aux événements de $Unstable(Y/_k/_\sim)$. Alors, l'algorithme 5.2, qui passe de $Unstable(Y/_{k-1}/_\sim)$ à $Unstable(Y_k/_\sim)$ en produisant les macro-événements stables correspondants, a une complexité en $O(\alpha(n\alpha + n^2 + \beta))$.*

En pratique, le désordre de réception des messages sera réduit ou, en tout cas, borné. De même, dans la mesure où le système observé ne génère que des comportements finiment engendrés, α et β sont bornés, eux aussi. Dans ce cas, la complexité de l'algorithme d'abstraction présenté ci-dessus est indépendante de la taille de l'exécution à abstraire. Il peut donc être utilisé en pratique, pour abstraire à la volée l'exécution d'un système que l'on désire superviser.

5.1.5 Applications

Les différents algorithmes ont été mis en œuvre dans un prototype, disponible à l'adresse : `http://www.irisa.fr/distribcom/Personal_Pages/gazagnaire/`. La figure 5.2 montre un exemple d'abstraction produite à partir d'une exécution aléatoire. Les événements (à gauche) et macro-événements (à droite) sont représentés par des rectangles et sont étiquetés par un ensemble de couples, qui correspondent à un type et au nombre d'occurrences de celui-ci dans le (macro)-événement considéré : par exemple, l'étiquette $\{\}$ représente un macro-événement ne contenant pas d'événement typé et l'étiquette "$\{(1,2),(3,2)\}$" représente un macro-événement contenant quatre événements typés, deux par le type 1 et deux par le type 3. De plus, nous avons fixé $\mathcal{B}(t) = 2$ pour tous les types $t \in T$. Enfin, dans le prototype que nous avons développé, nous avons donné une horloge vectorielle "classique" aux macro-événements produits en sortie de notre algorithme. Cela permet d'envisager, par exemple, un processus d'abstraction hiérarchique.

Cependant, ce n'est pas l'approche que nous considérons dans la suite de ce chapitre.

En effet, d'une part, nous supposons, à partir de maintenant, que les macro-événements produits sont générés avec le détail des horloges vectorielles des éléments qu'ils agrègent : ainsi, il est possible de reconstruire le morceau d'ordre auquel ils correspondent.

D'autre part, nous supposons que nous avons un modèle du système sous-jacent, donné sous la forme d'un HMSC, qui décrit donc un système dont les entités communiquent uniquement par échange des messages. Dans ce cadre, si l'on suppose que des informations supplémentaires concernant le typage peuvent être échangées entre les différents capteurs, nous avons montré, à la fin de la partie 5.1.2, qu'il était possible d'instrumenter correctement un système pour que l'abstraction produite soit exactement le découpage en atomes défini dans [Hélouët 00b, Ahuja 90].

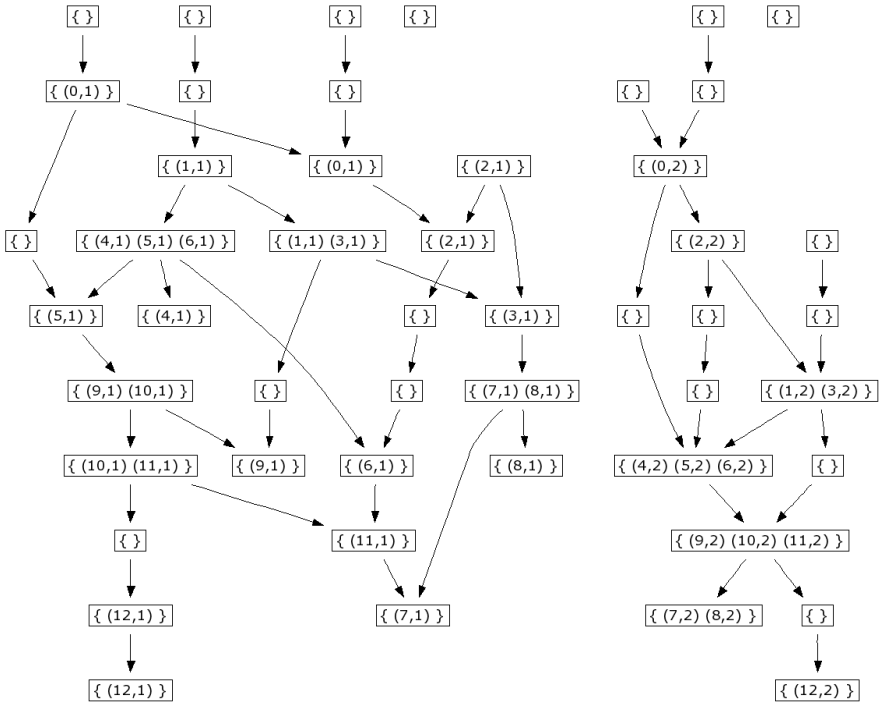


FIG. *5.2 – Un exemple d'abstraction produite à partir d'une exécution aléatoire.*

Ainsi, lorsque l'on considère que l'exécution du système est elle aussi un MSC (c'est-à-dire qu'on s'intéresse maintenant aux actions associées aux événements, plutôt qu'aux événements eux-mêmes), il est possible d'instrumenter un système de capteurs, afin que les macro-événements produits par l'abstraction de cette exécution soient exactement sa décomposition en atomes.

En combinant ces deux points, il est ainsi possible de définir un processus dans lequel les algorithmes décrits dans cette première partie vont servir à compter les différents atomes vus au cours de l'exécution. La technique que nous allons exposer dans la partie suivante consistera, alors, à déterminer si ce décompte est cohérent avec le modèle de haut niveau, donné en terme de HMSC.

5.2 Heuristique pour le problème de l'appartenance

Dans la partie précédente, nous avons présenté une technique permettant d'abstraire, et, donc, de compresser une exécution. De plus, nous avons montré que, lorsque l'observation a la forme d'un MSC, il est possible de transformer cette observation en un décompte de ses atomes. Nous supposons, dans cette seconde partie, que nous pouvons généraliser cette approche : étant donné une observation qui a la forme d'un pomset, il semble possible, en faisant des hypothèses adéquates, de calculer efficacement le décompte de ses pomsets premiers. Dans cette partie, nous montrons comment caractériser des modèles, donnés sous forme d'ex-

pression rationnelle de pomsets, qui permettent de décider efficacement si cette observation a pu être engendrée par l'une de ses exécutions.

Ce problème est un problème classique de la théorie des modèles formels, appelé *problème de l'appartenance*. Il consiste à décider si, étant donné un élément d'un monoïde et une expression rationnelle de ce monoïde, l'élément appartient au langage de l'expression rationnelle.

Avant de nous intéresser au cas des modèles de systèmes parallèles et répartis mixtes, nous rappelons les classes de complexité du problème de l'appartenance pour les différents modèles décrits dans le chapitre 1 :

- pour le monoïde libre, ce problème est NLOGSPACE-complet[Jones 75] ;
- pour le monoïde de traces, ce problème est NP-complet [Diekert 95] ;
- pour le monoïde des HMSC, ce problème est NP-complet [Alur 01].

Nous nous plaçons dans le cadre des rationnels de $\mathbb{P}(\Sigma, D)$, avec Σ, un ensemble d'actions et $D \subseteq \Sigma^2$, une relation quelconque.Dans ce cadre, le problème de l'appartenance est toujours un problème difficile : on peut montrer qu'il est lui aussi NP-complet. En effet, il est au moins aussi difficile que pour les HMSC et, de plus, vérifier si deux pomsets sont équivalents (ce que l'on veut tester pour vérifier une solution) peut se réduire à un problème d'isomorphisme de graphe étiqueté, problème qui est dans NP (la caractérisation de la classe de complexité du problème d'isomorphisme de graphe étiqueté reste un problème ouvert).

Ainsi, dans le cas où la taille de l'élément dont on veut tester l'appartenance devient importante, cette grande complexité pose un réel problème. En effet, en pratique, l'observation d'un système parallèle et réparti génère, avant filtrage, des fichiers qui contiennent des milliards d'entrées (voir le chapitre 6, pour un contexte plus détaillé concernant le diagnostic).

Nous proposons, dans cette partie, une heuristique au problème de l'appartenance lorsque le modèle considéré est une expression rationnelle de pomsets et l'observation, un décompte des différents pomsets premiers observés. Les outils que nous employons se basent sur les vecteurs de Parikh et sur l'utilisation d'outils matriciels.

L'avantage de cette heuristique est bien évidemment le gain en complexité spatiale (d'un facteur logarithmique) pour le stockage, et le gain de complexité temporelle pour la réponse à la question de l'appartenance. L'inconvénient est le même que pour toute méthode heuristique : la précision de cette méthode n'est pas exacte. Nous serons sûrs de la réponse donnée par notre méthode uniquement dans le cas où celle-ci est négative.

Enfin, nous donnerons une méthode pour quantifier le nombre d'exécutions, dans un modèle donné, qui possèdent le même décompte de pomsets premiers qu'une exécution observée. Cette méthode va dépendre exclusivement de la structure du modèle. Nous serons donc capables d'exhiber des modèles pour lesquels le problème de l'appartenance peut se résoudre en temps polynomial.

Approches similaires

D'une part, une formulation comparable de ce problème, utilisant le formalisme de la programmation linéaire et entière, a été effectuée par van der Aalst dans [vanderAalst 06]. Dans ces travaux, les modèles sont des classes très restreintes de réseau de Petri. De plus, la

complexité de ces problèmes reste NP-complet, puisque se basant sur des techniques d'optimisation linéaire.

D'autre part, le problème de l'appartenance d'un vecteur à un ensemble semi-linéaire est un problème équivalent à l'accessibilité d'un état pour un réseau de Petri. Ce problème a été très étudié et plusieurs classes de réseaux ont été identifiées pour lesquelles l'accessibilité est décidable avec une complexité polynomiale : citons, par exemple, les réseaux bornés sans conflits [Howell 88], les réseaux marqués [Commoner 71, Desel 95]. Notre approche permet de caractériser de nouvelles classes de modèles où la résolution de ce problème devient efficace.

5.2.1 Notations et définitions

Etant donné un ensemble $\Gamma = \{a_1, \ldots, a_n\}$, celui-ci est totalement ordonné par la relation d'ordre suivante : $a_i < a_j$ si, et seulement si, $i < j$. De plus, $\mathbb{N}$ désigne l'ensemble des entiers naturels.

Nous introduisons, maintenant, les notations de Parikh ([Parikh 66]) pour compter les lettres d'un mot. Soit a, une lettre. Le nombre d'occurrences de a dans $u \in \Gamma^\star$ est noté $|u|_a$. La fonction de Parikh $\psi : \Gamma^\star \to \mathbb{N}^{|\Gamma|}$ est définie par le vecteur $\psi(u) = (|u|_{a_1}, \ldots, |u|_{a_{|\Gamma|}})$. Le vecteur de Parikh d'un mot u est le vecteur $\psi(u)$. Notons que la fonction de Parikh est un morphisme du monoïde libre $\Gamma^\star$ vers le monoïde $(\mathbb{N}, +, 0^{|\Gamma|})$. Soit α, une expression rationnelle de $\Gamma^\star$. Nous étendons naturellement les définitions précédentes aux langages. L'image de Parikh de α, notée $\psi(\alpha)$ est l'ensemble $\{\psi(u) \mid u \in \mathcal{L}_{\Gamma^\star}(\alpha)\}$ des images de Parikh des mots du langage de α.

Tout d'abord, Parikh a montré que l'image de Parikh de n'importe quel langage algébrique peut se représenter à l'aide d'un ensemble semi-linéaire. C'est à dire qu'il existe une union finie d'équations linéaires de $\mathbb{N}^{|\Gamma|}$ qui décrit ses éléments. Or, tester l'appartenance d'un vecteur à un ensemble semi-linéaire est décidable ([Reutenauer 90, Mayr 81]).

Nous nous intéressons maintenant à l'approximation de modèles de pomsets donnée sous la forme de l'image de Parikh de ses représentants. Plus précisément, nous considérons comme représentant d'une expression rationnelle de pomsets, l'expression rationnelle des noms de pomsets premiers équivalente (ce qui correspond à l'image inverse du morphisme η, défini dans le thèorème 3.6 du chapitre 3).

Nous obtenons, par ce biais, une méthode pour tester si l'observation du décompte d'atomes n'appartient pas au langage. C'est cette idée que nous reprenons de manière constructive, en donnant une mesure, dépendant du modèle, qui va être corrélée à la complexité de la phase du test d'appartenance d'un vecteur à l'ensemble semi-linéaire que approxime ce modèle. *La méthode que nous proposons permet donc de faire de la vérification dans des cas où les modèles de sont pas reconnaissables.*

Plus précisément, nous allons nous poser le problème suivant :
Entrées :
- un modèle α, exprimé à l'aide d'une expression corationnelle de $\mathbb{P}(\Sigma, D)$;
- une observation o, donnée sous la forme d'un vecteur de $\mathbb{N}^{|\Gamma|}$;

Sortie : Est-ce qu'il existe $p \in \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha)$ tel que $\psi(\eta^{-1}(p)) = o$, où $\eta^{-1}(p)$ est la décomposition du pomset p en la trace de ses pomsets premiers[1] ?

[1] η est défini dans le théorème 3.6.

5.2.2 Vecteurs et matrices d'appartenance

L'idée que nous allons exploiter pour résoudre le problème d'appartenance est de le traduire dans un cadre matriciel, vers une résolution d'un système d'équations entières. Nous utiliserons, ensuite, un raisonnement algébrique pour caractériser les modèles sur lesquels le test d'appartenance peut se faire rapidement.

Pour cela, étant donné un ensemble de pomsets premiers Γ que l'on peut totalement ordonner et une expression rationnelle α construite à partir de Γ, considérons le $\Gamma^\star$-automate $\mathcal{A}_\alpha$, qui lui est structurellement équivalent. A partir de cet automate, nous construisons une matrice particulière, que nous avons appelé *matrice d'appartenance*, qui va nous permettre de résoudre le problème de l'appartenance de manière plus simple.

Définition 5.11 (matrice d'appartenance) *Soient $\Gamma \subset \mathbb{P}(\Sigma, D)$, un ensemble fini de pomsets premiers, α, une expression rationnelle de $\langle\Gamma\rangle_{\mathbb{P}(\Sigma,D)}$ et $\mathcal{A}_\alpha = (S, \rightarrow, \Gamma, S_i, S_f)$, le $\Gamma^\star$-automate structurellement équivalent à α.*

La matrice d'appartenance *de α est une matrice de taille $(n+|\Gamma|)\times m$, où n est le nombre d'états de $\mathcal{A}_\alpha$ et m son nombre de transitions, définie de la manière suivante :*

- *pour tout $(i,j) \in \{1\dots n\} \times \{1\dots m\}$, $M_\alpha[i,j] = 1$ si la j-ième transition arrive dans le i-ième état, -1 si elle en part, 0 sinon. Si la j-ième transition est une auto-boucle sur le i-ième état, alors $M_\alpha[i,j] = 0$;*
- *pour tout $(i,j) \in \{n+1\dots n+|\Gamma|\} \times \{1\dots m\}$, $M_\alpha[i,j] = 1$ si la j-ième transition est étiquetée par la $(i-n)$-ième lettre de Γ, 0 sinon.*

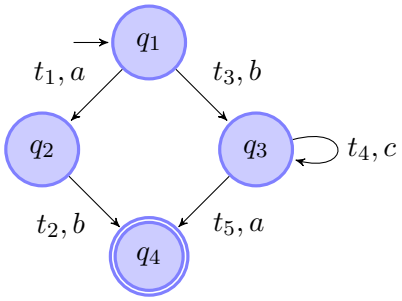
La première partie de cette définition fait clairement écho à la définition classique de la matrice d'adjacence d'un graphe orienté. La seconde partie de cette définition est plus originale et traduit l'étiquetage de ce graphe par Γ. Plus précisément, étant donné un chemin dans l'automate $\mathcal{A}_\alpha$, la seconde partie de la définition va nous servir à calculer le vecteur de Parikh du mot étiquetant ce chemin. Nous verrons cela, plus en détail, par la suite.

La figure 5.3 donne un exemple d'un automate et de la matrice d'appartenance qui lui est associée. Le modèle lié à cet automate est le rationnel $ab + bc^\star a$, a, b et c étant dans notre cas, des pomsets premiers dont le contenu n'a pas été précisé ici, et qui peuvent commuter, par exemple : $a \parallel b$.

De même que nous avons transformé un automate $\mathcal{A}_\alpha$ en une matrice de $\mathbb{N}^{(n+|\Gamma|)\times m}$, nous nous intéressons maintenant à la traduction d'un chemin de l'automate $\mathcal{A}_\alpha$ vers un objet dans $\mathbb{N}^{n+|\Gamma|}$. Pour cela, nous définissons le *vecteur d'appartenance* d'un chemin acceptant de la manière suivante :

Définition 5.12 (vecteur d'appartenance) *Soient $\Gamma \subset \mathbb{P}(\Sigma, D)$, un ensemble fini de pomsets premiers, α, une expression rationnelle de $\langle\Gamma\rangle_{\mathbb{P}(\Sigma,D)}$, $\mathcal{A}_\alpha$, le $\Gamma^\star$-automate structurellement équivalent à α ayant n états, $s, f \in \{1\dots n\}$, deux entiers et $u \in \Gamma^\star$.*

Le vecteur d'appartenance *$V_\alpha(s,f,u)$ de u à α, pour les états q_s et q_f, est un vecteur de $\mathbb{N}^{n+|\Gamma|}$ défini de la manière suivante :*



$$\begin{array}{c} \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} -1 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

FIG. *5.3 – Exemple de matrice d'appartenance.*

- *si* $s \neq f$ *:* $V_\alpha(s,f,u)[s] = -1$ *et* $V_\alpha(s,f,u)[f] = 1$ *;*
- *si* $s = f$ *:* $V_\alpha(s,f,u)[s] = V_\alpha(s,f,u)[f] = 0$ *;*
- *pour tout* $i \in \{1 \dots n\} - \{s,f\}$, $V_\alpha(s,f,u)[i] = 0$ *;*
- *pour tout* $i \in \{1 \dots |\Gamma|\}$, $V_\alpha(s,f,u)[n+i] = \psi(u)[i]$.

Cette définition de vecteur d'appartenance est une extension du vecteur de Parikh classique (en effet, on retrouve le vecteur de Parikh dans les $n-|\Gamma|$ dernières lignes de ce vecteur). Cependant, le vecteur d'appartenance $V_\alpha(s,f,u)$ correspond intuitivement aux chemins de α ayant pour état initial s et état final f et étiquetés par un mot v tel que $\psi(u) = \psi(v)$.

Par exemple, si l'on prend comme modèle $\alpha = ab + bc^\star a$, dont l'automate structurellement équivalent est donné dans la figure 5.3, le chemin $\rho = q_1 \xrightarrow{b} q_3 \xrightarrow{c} q_3 \xrightarrow{c} q_3 \xrightarrow{a} q_4$ correspondra à un vecteur d'appartenance $V_\alpha(1,4,bcca) = (-1,0,0,1,1,2,1)$: q_1 est l'état initial de ρ, q_4 son état final et $\psi(bcca) = (1,2,1)$.

Finalement, nous donnons une dernière définition inspirée de la théorie des graphes. Nous utilisons des vecteurs de $\mathbb{N}^m$ pour représenter le support d'une exécution, associé à un chemin dans un automate. Plus précisément, chaque ligne de ce vecteur est associée à une des transitions de l'automate et contient le nombre de fois que cette transition a été tirée dans le chemin considéré.

Définition 5.13 (connexité) *Soient* $\Gamma \subset \mathbb{P}(\Sigma, D)$*, un ensemble fini de pomsets premiers,* α*, une expression rationnelle de* $\langle\Gamma\rangle_{\mathbb{P}(\Sigma,D)}$*,* $\mathcal{A}_\alpha$*, le* $\Gamma^\star$*-automate structurellement équivalent à* α *et* X*, un vecteur de* $\mathbb{N}^m$ *(avec* m *le nombre de transitions de* $\mathcal{A}_\alpha$*).*

Nous dirons que X *est* α-connexe *si le graphe* (S,T) *est connexe, avec :*
- S *est l'ensemble des états de* $\mathcal{A}_\alpha$ *;*
- $T = \{(s,t) \mid \exists i,\ s \xrightarrow{a} t$ *est la* i^e *transition de* $\mathcal{A}_\alpha$ *et* $X[i] \neq 0\}$.

Par exemple, en considérant l'automate $\mathcal{A}_\alpha$, défini dans la figure 5.3, le vecteur $(2,0,0,5,3)$ n'est pas α-connexe car le graphe dont les états sont q_1, q_2, q_3 et q_4 et dont les arêtes sont les transitions t_1, t_4 et t_5, n'est pas connexe.

5.2.3 Résultat d'approximation

Nous avons donc vu les définitions principales, qui vont nous permettre de plonger notre problème d'appartenance dans un cadre algébrique. Plus précisément, nous allons montrer que, répondre à la question : "est-ce qu'il existe un mot dans le langage de notre modèle qui a le même vecteur de Parikh que notre observation ?" est équivalent à trouver s'il existe un chemin dans l'automate associé à notre modèle qui a le même vecteur d'appartenance. Plus précisément, le théorème 5.14 donne une formulation matricielle à ce problème.

Théorème 5.14 *Soient $\Gamma \subset \mathbb{P}(\Sigma, D)$, un ensemble fini de pomsets premiers, α, une expression rationnelle de $\langle\Gamma\rangle_{\mathbb{P}(\Sigma,D)}$, M_α, la matrice d'appartenance de α dans $\mathbb{N}^{(n+|\Gamma|)\times m}$ et u, un mot de $\Gamma^\star$. Alors :*

$$\textit{Il existe un chemin acceptant de } \mathcal{A}_\alpha \textit{ étiqueté par } v \in \Gamma^\star \textit{ tel que } \psi(u) = \psi(v)$$

$$\Leftrightarrow$$

$$\exists X_0 \in \mathbb{N}^m \ \alpha\textit{-connexe et } i, j \in \{1 \dots n\} \textit{ tel que } M_\alpha \cdot X_0 = V_\alpha(i, j, \psi(u))$$

Preuve: D'une part, supposons qu'il existe un pomset v qui soit reconnu par α, tel que $\psi(\eta(v)) = \psi(\eta(o))$. Il existe donc un chemin acceptant ρ dans $\mathcal{A}_\alpha$ étiqueté par v. Nommons q_s, son état initial et q_f, son état final. On peut associer à ce chemin ρ, un vecteur X_ρ de $\mathbb{N}^m$, qui compte le nombre de transitions tirées par ρ : $X_\rho[i] =$"nombre de fois que la i-ième transition de $\mathcal{A}_\alpha$ est tirée dans ρ".

Or, ρ étant un parcours dans le graphe direct associé à l'automate $\mathcal{A}_\alpha$, démarrant dans l'état q_s et finissant dans l'état q_f, d'après la loi d'Euler (appelée aussi première loi de Kirchnoff, ou équation de conservation de flot), pour chaque état q différent de q_s et q_f de $\mathcal{A}_\alpha$, on a, en notant $q \xrightarrow{*} *$, l'ensemble des transitions qui sortent de q, $* \xrightarrow{*} q$, l'ensemble des transitions qui y entrent et $* \xrightarrow{a} *$ l'ensemble des transitions étiquetées par a :

$$\sum_{t_i \in * \xrightarrow{*} q} X_\rho[i] = \sum_{t_j \in q \xrightarrow{*} *} X_\rho[j]$$

et X_ρ est connexe. De plus, d'après ces mêmes lois, si $q_f \neq q_s$, alors :

$$1 + \sum_{t_i \in * \xrightarrow{*} q_s} X_\rho[i] = \sum_{t_j \in q_s \xrightarrow{*} *} X_\rho[j]$$

$$\sum_{t_i \in * \xrightarrow{*} q_f} X_\rho[i] = 1 + \sum_{t_j \in q_f \xrightarrow{*} *} X_\rho[j]$$

et si $q_f = q_s$, les premières équations sont valides pour ces états, aussi.

De plus, par définition du vecteur de Parikh :

$$\sum_{t_i \in * \xrightarrow{a_j} *} X_\rho[i] = \psi(v)[j]$$

et donc :

$$\sum_{t_i \in * \xrightarrow{a_j} *} X_\rho[i] = \psi(u)[j]$$

Au final, nous avons donc : $M_\alpha \cdot X_\rho = V_\alpha(s, f, u)$.

D'autre part, montrons la réciproque. Supposons donc qu'il existe une solution entière positive et connexe à l'équation. Cette solution nous donne l'existence d'un parcours dans l'automate, démarrant dans q_i et terminant dans q_j (si $i = j$, la solution est un cycle). De plus, le vecteur de Parikh de ce parcours (c'est-à-dire le décompte des pomsets premiers vus le long de ce chemin) est exactement $\psi(\eta(o))$. Il existe donc un mot v, étiquetant ce chemin, qui est tel que $\psi(\eta(o)) = \psi(\eta(v))$. □

Ce théorème nous donne une nouvelle formulation du problème initial, donnée en terme de problème matriciel. Il est alors possible d'utiliser des outils de programmation linéaire et entière pour résoudre ce nouveau problème. Cependant, la complexité du problème est toujours grande : le nouveau problème est toujours NP-complet, et nous avons perdu de la précision, puisque, même si nous sommes capables de dire si un mot n'appartient pas à un langage, plusieurs mots peuvent avoir le même vecteur de Parikh, dont un qui est dans le langage, et l'autre non. Cependant, cette méthode nous permet d'appliquer des outils algébriques pour trouver des cas où nous aurons, en même temps, efficacité et précision (concernant le chemin emprunté).

Reprenons l'exemple de la figure 5.3. Supposons que nous ayons observé un vecteur de Parikh de $(1, 1, 1)$ (c'est-à-dire que l'exécution observée est composée d'un a, d'un b et d'un c). Le système à résoudre est alors le suivant :

$$\begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 \end{matrix} \\ \begin{pmatrix} -1 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \cdot X = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

En utilisant le pivot de Gauss, nous pouvons nous ramener au système d'équations suivant :

$$\begin{array}{c} \begin{matrix} t_4 & t_5 & t_3 & t_2 & t_1 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \cdot X = \begin{pmatrix} 1 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Dans ce système, le seul paramètre libre est t_1, et il peut prendre deux valeurs 0 ou 1. Seul, le cas $t_1 = 0$ donne une solution entière positive α-connexe. Cette solution donne aussi $t_1 = t_2 = 0$ et $t_3 = t_4 = t_5 = 1$ et nous permet donc d'identifier le support du seul chemin possible $q_1 \xrightarrow{b} q_3 \xrightarrow{c} q_3 \xrightarrow{a} q_4$.

Cet exemple nous permet de voir que, plus le nombre de paramètres libres est petit dans le système d'équations que nous donne le théorème 5.14, plus l'espace des solutions sera petit et donc rapide à explorer. Cet indice, que nous allons appeler *indice de Parikh* du modèle α, est donc un indicateur précieux sur l'efficacité de notre technique.

Définition 5.15 (Indice de Parikh) *Soient $\Gamma \subset \mathbb{P}(\Sigma, D)$, un ensemble fini de pomsets premiers, α, une expression rationnelle de $\langle\Gamma\rangle_{\mathbb{P}(\Sigma,D)}$ et $\mathcal{A}_\alpha$, le $\Gamma^\star$-automate structurellement équivalent à α.*

*L'*indice de Parikh *du modèle α, noté $P(\alpha)$, correspond au nombre de paramètres libres dans l'équation $M_\alpha \cdot X = V$, pour tout V. En posant $rank(M)$, le rang de la matrice M et m, le nombre de transitions dans $\mathcal{A}_\alpha$, cet indice est défini par :*

$$P(\alpha) = m - rank(M_\alpha)$$

L'indice de Parikh d'un modèle correspond à la difficulté à trouver un chemin dans ce modèle ayant le même vecteur de Parikh que toute observation donnée. En particulier, pour les modèles dont l'indice de Parikh est 0, savoir si un support connexe ayant le même vecteur de Parikh existe dans l'automate correspondant au modèle, est immédiat : en effet, il suffit d'inverser M_α (ce qui peut être fait une fois, au tout début de l'analyse) et de calculer $M_\alpha^{-1} \cdot V_\alpha(s, f, \eta(o))$, pour tous les états initiaux s et états finaux f de $\mathcal{A}_\alpha$. Il suffit alors de vérifier si le vecteur obtenu est α-connexe pour obtenir directement la réponse à notre problème.

Au final, nous disposons donc d'une mesure sur les modèles qui permet, quand celle-ci vaut 0, de caractériser des systèmes où la complexité temporelle du diagnostic approximé (c.a.d. qui ne prend en compte que les supports des exécutions) *ne dépend pas de la taille de l'observation*. Ce type de modèle semble donc très intéressant en pratique.

5.3 Conclusions et perspectives

Résumé

Dans ce chapitre, nous avons présenté deux techniques efficaces qui permettent de traiter des fichiers logs volumineux.

La première technique est une méthode d'abstraction d'exécution répartie qui permet de produire, à la volée, une compression de l'observation sous forme de macro-événements. Notre contribution est à situer principalement en continuité des travaux de [Basten 97] et [Hélouët 00b]. [Basten 97] a introduit la notion de précédence faible et son codage vectoriel. [Hélouët 00b] a défini la notion de décomposition atomique dans les "Message Sequence Charts" et proposé un algorithme hors-ligne fondé sur le calcul de composantes fortement connexes d'un graphe. Nous améliorons ces travaux sur les points suivants.

D'une part, nos algorithmes marchent à la volée sur un observateur implanté dans un processus du système. Gérer les événements qui arrivent dans le mauvais ordre demande un soin particulier (notion de macro-événement en cours de formation et détection de la stabilité).

D'autre part, la construction de nos macro-événements est paramétrée par une fonction de typage, et par l'obligation que nous nous donnons d'avoir un ordre partiel clos par types. Cela présente une généralisation intéressante, aussi bien au niveau théorique que pratique.

Finalement, le fait de rester strictement dans le cas d'ordres partiels permet de reproduire le schéma d'abstraction à plusieurs niveaux, offrant ainsi une fonction d'abstraction hiérarchique. L'observateur agit comme un filtre sur le système produisant les événements de plus bas niveau. On donc peut envisager une abstraction hiérarchique ne donnant comme entrée à un observateur d'une couche donnée les sorties produites par les algorithmes de compressions de la couche précédente.

La seconde technique que nous avons présentée consiste à faire une approximation de diagnostic, en utilisant des méthodes algébriques. Nous avons aussi caractérisé les modèles pour lesquels cette méthode d'approximation possède une efficacité qui ne dépend pas de la taille de l'observation. L'approximation considérée s'appuie sur la construction de l'image de Parikh de l'observation. De plus, la méthode de résolution s'appuie sur les équations de flot dans un graphe orienté, afin de déterminer les chemins étiquetés par le bon vecteur de Parikh dans l'automate associé au modèle considéré. Cette approche permet de caractériser une nouvelle classe de modèles pour lesquels l'appartenance à leur image de Parikh peut se décider en temps polynomial : les modèles où l'indice de Parikh est égal à 0.

Perspectives

Les perspectives ouvertes peuvent s'organiser comme le présent chapitre : d'une part, des perspectives concernant la compression d'exécutions réparties et d'autre part, des perspectives concernant la mise au point d'un test d'appartenance efficace.

Les premières perspectives concernent la compression de l'observation d'une exécution d'un système parallèle et réparti.

Tout d'abord, la technique que nous avons développée produit, à nouveau, un ordre partiel de macro-événements. Il serait donc intéressant de concevoir une abstraction hierarchique, afin d'augmenter le facteur de compression. Il faudrait aussi étudier comment répartir cette abstraction, en mettant au point une architecture d'abstraction et de compression répartie.

Ensuite, il serait intéressant de regarder comment intégrer cette approche avec les techniques de compression d'ordres partiels développées, par exemple, dans [Alur 03, Goel 03], qui adaptent, aux ordres partiels étiquetés, des algorithmes classiques de compression de chaînes de caractères.

Enfin, un dernier point à étudier est la généralisation du découpage en atomes par le processus d'abstraction. En effet, en choisissant correctement la fonction de typage qui paramètre l'abstraction, nous avons vu qu'il était possible de découper une exécution ayant la forme d'un MSC en sa décomposition en atomes. Quels sont les restrictions à apporter pour que cette construction fonctionne aussi dans le cadre des HMSC causaux ? des pomsets premiers ?

Les secondes perspectives concernent la mise au point d'un test d'appartenance efficace.

Tout d'abord, il serait intéressant de caractériser syntaxiquement les modèles ayant un indice de Parikh égal à 0. Est-ce que ces modèles sont reliés, d'une quelconque façon, aux modèles pour lesquels on sait déjà que le test d'appartenance à l'image de Parikh peut se faire efficacement ?

Ensuite, il nous parait intéressant d'utiliser cet indice pour aider à la fabrication de systèmes plus facilement observables. En effet, étant donné un modèle d'indice de Parikh donné, il semble possible de déterminer quelles sont les modifications à apporter pour que son indice de Parikh diminue. Ces modifications prendraient la forme d'observateurs supplémentaires à ajouter dans le système, par exemple.

Enfin, il parait important d'améliorer la précision de cette méthode : en effet, nous ne sommes capables d'identifier que des chemins probables dans le modèle initial, chemins qui peuvent avoir engendré l'exécution observée. Cependant, en augmentant l'observation pour permettre de compter les différents sous-mots déjà observés, la précision peut être améliorée. Il faudrait alors adapter nos méthodes aux matrices de Parikh, définies par Mateescu dans [Mateescu 04], qui permettent, justement, de compter les sous-mots d'une observation plutôt que ses lettres.

Chapitre 6

Supervision avec observation partielle

Nous avons vu, dans la partie I de ce document, différents modèles pour spécifier des systèmes parallèles et répartis et nous avons introduit un modèle mixte, localement asynchrone et globalement communicant qui les généralise. Ensuite, la partie II s'est attachée à décrire des techniques de vérification de tels modèles.

Ce chapitre constitue, avec le chapitre précédent, la partie III de cette thèse, consacrée à la *supervision*. Rappelons que dans le chapitre précédent, nous nous sommes intéressés à la supervision avec observation complète, avec l'objectif de fournir des méthodes efficaces d'analyse.

Dans ce chapitre, nous allons maintenant nous intéresser aux analyses de supervision qui peuvent s'employer dans le cas où nous n'avons qu'une connaissance *partielle* de l'exécution observée. Le but des techniques que nous développons dans ce chapitre est d'inférer, à partir d'un modèle complet du système observé, certains éléments qui n'ont pas été observés.

Plus précisément, ce chapitre s'intéresse à deux problèmes spécifiques qui exploitent la sémantique d'ordres partiels des modèles manipulés et qui permettent d'analyser des systèmes dont le comportement n'est pas borné. La première technique, appelée diagnostic, consiste à inférer les *événements non observés* dans une observation à l'aide d'un modèle du système. Elle consiste à construire un nouveau modèle complet dont la projection de toute exécution sur les événements observables explique l'observation. La seconde technique, appelée corrélation d'événements, consiste à inférer les *causalités non observées* dans une observation, à l'aide d'un modèle du système. Elle consiste à rajouter à l'observation, des causalités qui apparaissent dans la projection de toutes les exécutions du modèle.

Contexte

Les techniques de supervision ont de nombreuses applications, et plus particulièrement l'aide à l'interprétation de fichiers logs volumineux (voir l'introduction du chapitre 5 pour plus de détails).

Cependant, dans ce chapitre et contrairement au chapitre précédent, nous considérons qu'un premier pré-traitement de filtrage des logs a été réalisé, afin de ne retenir qu'une petite partie des alarmes enregistrées. Ce filtrage permet d'obtenir des structures de données de taille raisonnable, et donc des algorithmes qui peuvent s'exécuter en des temps réalistes. Ce pré-traitement, couplé avec le fait qu'il n'est pas possible d'observer tous les événements qui se déroulent dans un système (car certaines parties de ce système ne sont que partiellement instrumentées, par exemple), nous forcent à ne considérer que des *observations partielles*, c'est-à-dire des exécutions dont certains événements n'ont pas été observés.

Dans ce contexte, différents problèmes ont été étudiés ces dernières années. Nous en évoquons, dans la suite, un bref aperçu, ainsi que des solutions théoriques ou techniques apportées.

Diagnostic de fautes

Le premier type de diagnostic qui peut être défini, est le diagnostic de fautes. Pour ce problème, sont fixés un ensemble de fautes possibles et un modèle d'exécution qui indique les comportements du système observé et à quel moment peuvent se produire ces fautes. Ensuite, étant donné une exécution partiellement observée (en particulier les fautes ne peuvent pas être observées), la question à se poser est alors la suivante : est-ce qu'il existe une exécution du modèle contenant une faute possible qui puisse expliquer cette observation, et si oui, quelle est cette faute ? De plus, on veut pouvoir répondre à cette question en ligne, avec un délai de n événements observés. Sampath et al. ont montré dans [Sampath 96] que ce problème est décidable lorsque le modèle est donné sous la forme d'un automate. Ils ont, pour cela, construit un *diagnostiqueur* qui correspond à une projection du modèle du système sur les événements observables, dans laquelle est gardée une mémoire de n événements afin de savoir s'ils correspondent à des fautes. Lorsque, pour chaque séquence d'événements observables, il est possible de trouver (au moins) deux séquences dans le modèle du système, l'une dont l'un des n derniers événements correspond à une faute, l'autre dont les n événements correspondent à un comportement normal, alors le système n'est pas diagnosticable. Dans le cas contraire, toutes les séquences sont soit correctes, soit fautives. Cela veut dire qu'une faute peut être détectée à coup sûr après un certain délai.

Jéron et al. décrivent dans [Jéron 06] une approche similaire pour des modèles de fautes plus complets. Les mêmes techniques peuvent être utilisées pour des modèles plus complexes, tant que ceux-ci restent stables par projection (ce qui est le cas pour les réseaux de Petri bornés, par exemple, mais pas pour les réseaux de Petri quelconques, ni pour les HMSC). Lorsque la construction d'un diagnostiqueur de taille finie est impossible, il faut se restreindre à construire une structure dont la taille n'est pas, a priori, bornée, comme c'est le cas pour les techniques suivantes.

Reconstruction d'explications

Une deuxième famille de techniques de diagnostic consiste, à partir d'un modèle du système observé et d'une exécution partielle, à construire un générateur de toutes les explications possibles de cette exécution. L'idée directrice de cette approche consiste à déplier le modèle initial en se restreignant aux explications possibles de l'observation. Ensuite, à partir de ce modèle générateur, il est possible de vérifier, par exemple, qu'un événement fautif a eu lieu, ou non. La taille du modèle générateur dépend, en général, de la taille de l'observation, et n'est donc pas bornée.

Dans [Benveniste 03], Benveniste et al. s'intéressent à une instance de ce problème où le modèle du système est exprimé à l'aide de réseaux de Petri bornés, dans lesquels tous les cycles contiennent au moins un événement observable.

Nous avons montré, dans [Hélouët 06], que cette approche peut aussi s'étendre aux HMSC, sans aucune restriction structurelle sur les modèles. Plus précisément, nous avons montré que, pour tout modèle exprimé à l'aide d'un HMSC et d'une explication donnée sous forme de MSC, nous pouvions construire un générateur fini de toutes les explications de l'observation, donné lui aussi sous forme de HMSC. L'approche que nous décrivons dans ce chapitre est plus générale que celle donnée dans [Hélouët 06], que nous ne détaillerons pas dans ce manuscrit.

Corrélation d'événements

Enfin, une dernière famille de techniques de diagnostic consiste à annoter une observation à l'aide d'un modèle du système, afin de donner des indications sur ce qui a pu se passer pendant l'exécution observée, et permet de simplifier son interprétation. Dans les systèmes classiques de diagnostic déployés dans l'industrie, ce sont le plus souvent des règles simples qui sont appliquées. Parmi celles-ci, nous pouvons citer :

- des règles de compression : si plusieurs occurrences d'un même événement apparaissent, alors il est possible, dans certains cas, d'enlever les événements redondants ;
- des règles d'énumération : plutôt que de supprimer les éléments redondants, on compte le nombre d'occurrences de chacun, et on remplace les occurrences d'un événement par le nombre de fois où il apparaît (partie 5.2) ;
- des règles de suppression fondées sur des priorités : si une alarme de faible priorité est observée, suivie d'une alarme de plus forte priorité, alors il est possible de supprimer celle de priorité la plus faible ;
- des règles de généralisation : à partir de l'observation et de règles très simples de génération de nouveaux événements, on construit une nouvelle observation de plus haut niveau ([Dousson 99] et partie 5.1) ;
- des règles de corrélation qui établissent des liens de cause à effet entre les événements observés [Nygate 95].

Toutes ces règles sont utilisées dans des systèmes experts qui lisent et simplifient l'enregistrement d'une exécution observée.

Dans ce chapitre, nous considérons que la corrélation d'événements consiste à inférer des causalités entre les événements, ce qui peut aussi se décliner en la recherche de la cause initiale d'une panne. En effet, reconstruire les causalités permet de réduire le nombre d'éléments minimaux, qui sont, a priori, la ou les causes du phénomène observé.

Organisation du chapitre

Ce chapitre est organisé de la manière suivante.

Tout d'abord, nous donnons, dans la partie 6.1, la modélisation de l'architecture d'observation partielle que nous avons retenue pour développer nos théories du diagnostic centralisé et réparti. Nous donnerons, ainsi, la formulation de deux problèmes d'inférence que nous résolvons ensuite dans les parties suivantes.

Nous traitons, d'une part, dans la partie 6.2 de l'inférence d'événements non observés dans un contexte centralisé et réparti. Cette technique permet d'engendrer, à l'aide d'un générateur fini, l'ensemble des exécutions complètes qui ont pu se produire, et dont seulement une partie a été observée.

Nous traitons, d'autre part, dans la partie 6.3, de l'inférence de causalités non observées, un problème plus connu sous le nom de corrélation d'événements. Ce problème permet de retrouver les causalités qui apparaissent dans toutes les exécutions complètes expliquant une observation partielle donnée, et donc de réduire le nombre de causes possibles d'un phénomène observé.

Enfin, la partie 6.4 conclut ce chapitre en donnant quelques perspectives.

6.1 Modèles d'observation

Nous posons, dans cette partie, les différents hypothèses et formalismes qui vont nous permettre d'apporter des méthodes rigoureuses de techniques de diagnostic et de corrélation d'événements. Tout d'abord, nous caractérisons les modèles d'architecture centralisés et répartis et les conséquences de l'utilisation de ces modèles, sur la définition d'une observation. Puis, dans ce cadre, nous définissons ce qu'est une explication, ainsi qu'une explication guidée par un modèle. Enfin, nous posons les problèmes théoriques à résoudre, ce que nous faisons dans la partie suivante de ce chapitre.

6.1.1 Observation et architectures

Nous décrivons maintenant les deux architectures que nous considérons pour la technique de diagnostic que nous allons présenter ici. Nous présentons tout d'abord une version centralisée de cette architecture, puis nous indiquerons les modifications à apporter pour travailler dans un cadre réparti.

Tout d'abord, rappelons que nous considérons que le système que nous observons est composé d'un ensemble d'agents indépendants, qui réagissent à leur environnement et qui communiquent avec leurs voisins, soit par mémoire partagée, soit par échange de messages. Ces agents, au fur et à mesure qu'ils exécutent le comportement pour lequel ils ont été programmés, produisent un certain nombre d'événements. Nous faisons l'hypothèse que le type de ces événements appartient à un ensemble fini, correspondant à un nombre fini d'actions possibles : par exemple, lire ou modifier une variable partagée donnée, lancer une procédure de synchronisation entre plusieurs agents, envoyer ou réceptionner un message contenant un entête donné, etc. Toutes ces actions forment un ensemble fini Σ d'étiquettes qui sont associées aux événements produits par le système.

Cadre centralisé

La figure 6.1 décrit schématiquement l'architecture considérée, qui est très proche d'une architecture de type SNMP. Chaque agent est équipé d'un capteur (représenté sur la figure par un double losange), qui envoie les événements qu'il observe à un observateur centralisé (représenté sur la figure par un cylindre). Les connexions entre les agents sont représentées par des lignes en pointillé. Le système d'enregistrement ("log system") reçoit les événements observables produits par les capteurs et les enregistre dans un fichier.

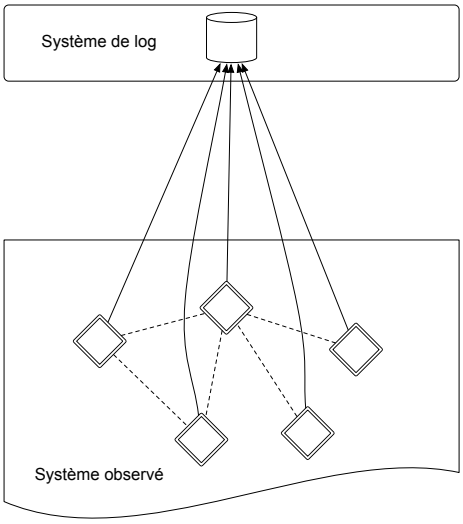


FIG. *6.1 – Architecture centralisée du mécanisme d'observation.*

Remarquons que les capteurs que nous ajoutons ne peuvent pas tout enregistrer. La première raison est que la taille d'un fichier d'enregistrement est nécessairement limitée, et qu'il faut donc faire des choix pour savoir ce qui doit être enregistré. La seconde raison est qu'un certain nombre d'agents ne peut pas être instrumenté, car il correspond, par exemple, à du matériel (comme une carte réseau), ou à une zone logicielle protégée (comme un noyau de systèmes d'exploitation). Nous modélisons ceci en disant que seulement un sous-ensemble de Σ est observable. Nous noterons ce sous-ensemble par Σ_o. De même, la causalité entre les différents événements observés ne peut pas être totalement observée et enregistrée. En effet, pouvoir observer toute la causalité d'un système nécessite l'utilisation d'outils qui peuvent ralentir les communications et nécessitent des droits d'accès privilégiés, tels que les horloges vectorielles de Fidge et Mattern ([Fidge 91, Mattern 89]). Ainsi, un fichier d'enregistrement peut être défini comme un LPO $o = (E_o, \leq_o, \lambda_o)$, où, $\lambda_o(E_o) \subseteq \Sigma_o$.

Nous montrerons, par la suite, qu'à partir d'un modèle approprié, décrit par une expression rationnelle de pomsets, nous pouvons retrouver les événements non observés (partie 6.2) ainsi que les causalités non observées (partie 6.3).

Cadre réparti

Il est maintenant très simple de généraliser le cadre précédent au cadre du diagnostic réparti. Pour cela, il suffit de considérer, comme l'exprime la figure 6.2, qu'il existe différents

systèmes de log qui observent diverses parties du système. Plus précisément, nous supposerons qu'il existe $\Sigma_o^1, \ldots, \Sigma_o^n$, des alphabets d'observations non nécessairement disjoints inclus dans Σ. Ainsi, une observation répartie sera un ensemble $\{o_1, \ldots, o_n\}$ de LPO $o_i = (E_o^i, \leq_o^i, \lambda_o^i)$, où, pour tout $i \in \{1 \ldots n\}$, $\lambda_o^i(E_o^i) \subseteq \Sigma_o^i$).

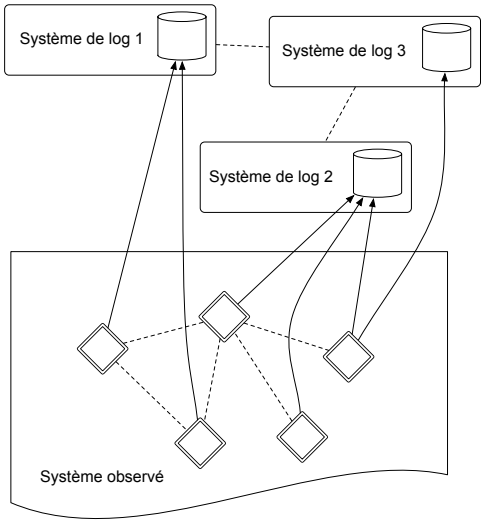


FIG. *6.2 – Architecture répartie du mécanisme d'observation.*

Les différents systèmes de log sont, eux mêmes, reliés par un réseau de communication, mais nous considérons qu'à ce niveau, l'information échangée est complète. Le défi sera alors d'être capable de combiner les différentes explications données par chaque système, afin d'obtenir une explication globale valide et complète. Nous montrerons, dans les parties 6.2 et 6.3 comment fusionner correctement ces informations dans le cadre du diagnostic et de la corrélation d'événements.

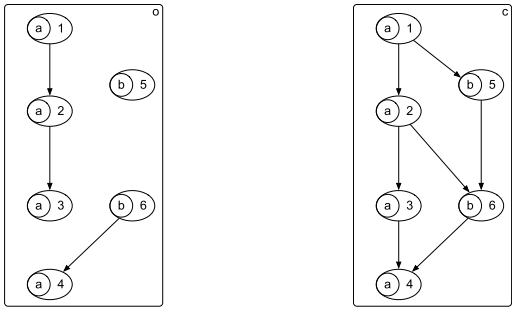
6.1.2 Observation et explications

Nous avons vu précédemment le modèle d'architecture considéré. Nous détaillons maintenant le lien entre, observation d'une part, et exécution d'un modèle d'autre part. Remarquons, en particulier, que l'opération de projection joue un rôle central dans les définitions que nous donnons, et que, par conséquent, les outils de projection exacte de modèles, développés dans le chapitre 4 de ce manuscrit, vont jouer dans ce chapitre un rôle important.

Nous commençons par donner la définition d'une *explication*.

Définition 6.1 (Explication) *Soient Σ, un ensemble d'actions et $\Sigma_o \subseteq \Sigma$, l'ensemble des actions observables. Une* explication *d'une observation o, qui est un* LPO $o = (E_o, \leq_o, \lambda_o)$, *est un* LPO $o' = (E_o, \leq, \lambda_o)$, *tel que $\leq$ soit une extension d'ordre de $\leq_o$, c'est-à-dire $\leq_o \subseteq \leq$. L'ensemble des explications de o est noté $[\![o]\!]$.*

Notons que o et ses explications sont définies sur le même ensemble d'événements E_o et ne diffèrent que par l'ordre partiel qui relie ces événements.



(a) Observation (b) $c \in [\![o]\!]$.

FIG. *6.3 – Une observation et une explication associée.*

La figure 6.3 donne une illustration de ces définitions. Plus précisément, la figure 6.3(a) donne une observation o, et la figure 6.3(b) donne une explication possible de cette observation, car la structure c qui y est définie, étend l'ordre de o.

Nous généralisons, ensuite, la définition précédente aux *explications guidées par un modèle*. A partir d'un modèle, donné sous la forme d'un rationnel de pomsets et d'une observation, nous voulons caractériser toutes les explications de cette observation qui ont pu être engendrées par ce modèle. Pour cela, rappelons que $[l]$ désigne le pomset qui représente la classe d'isomorphisme du LPO l. Nous avons donc :

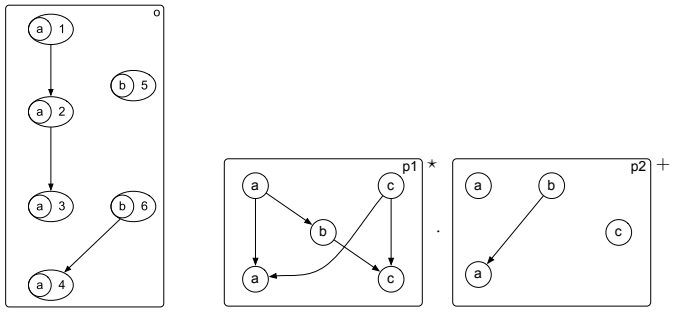
Définition 6.2 (Explication guidée par un modèle) *Soient Σ, un ensemble d'actions, $D \subseteq \Sigma^2$, une relation, $\Sigma_o \subseteq \Sigma$, l'ensemble des actions observables, et α, une expression rationnelle de $\mathbb{P}(\Sigma, D)$. L'ensemble des* explications de o, guidées par le modèle α *ou, plus simplement, les α-explications de o, est l'ensemble des explications de o dont la classe d'isomorphisme appartient à la projection du langage de α sur Σ_o. Plus formellement, l'ensemble des α-explications de o, noté $[\![o]\!]_{\Sigma_o,\alpha}$, est tel que $l \in [\![o]\!]_{\Sigma_o,\alpha}$ si, et seulement si, $[l] \in \pi_{\Sigma_o}(\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha))$ et $l \in [\![o]\!]$.*

La figure 6.4 illustre les définitions précédentes. Plus précisément, la figure 6.4(a) reprend l'observation de la figure 6.3(a), en lui ajoutant un modèle α défini dans la figure 6.4(b), avec $\Sigma = \{a, b, c\}$, et $D = \{(a, a), (c, b), (c, c)\}$. La figure 6.4(c) donne toutes les α-explications de o, qui correspondent à l'exécution de p_1 suivi de p_2 pour les explications e_1 et e_2, et à l'exécution de p_1 suivi à nouveau de p_1 pour l'explication e_3. Remarquons aussi que l'explication c de la figure 6.3(b) n'est pas une α-explication de o.

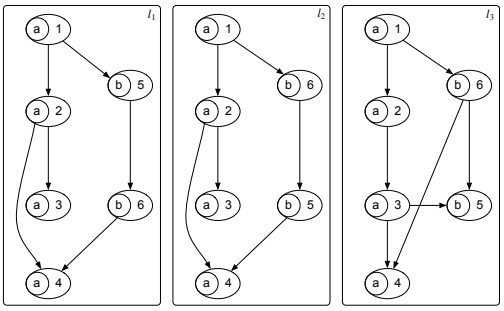
Toutes ces définitions se généralisent aisément aux pomsets. Nous noterons, ainsi, $[[\![o]\!]]$, l'ensemble des pomsets qui expliquent le LPO o, ainsi que $[[\![o]\!]_{\Sigma_o,\alpha}]$, l'ensemble des pomsets qui expliquent o, en étant guidé par le modèle α.

6.1.3 Observation et inférences

Nous définissons, maintenant, les deux problèmes d'inférence auxquels nous nous intéressons dans la suite de ce chapitre.



(a) Observation (b) $\alpha = p_1^\star p_2^+$, $D = \{(a,a),(c,b),(c,c)\}$.



(c) $[\![o]\!]_{\{a,b\},\alpha} = \{l_1, l_2, l_3\}$

FIG. *6.4 – Observations et explications guidées par un modèle.*

Le premier problème consiste à inférer les événements non observés. Comme cet ensemble d'événements n'est pas borné, la solution à apporter n'est évidemment pas une énumération de tous les événements qui peuvent être ajoutés. Au contraire, nous avons plutôt besoin de construire un générateur fini de toutes les explications. Plus précisément, le générateur que nous allons construire pour résoudre ce problème aura la forme d'une expression rationnelle du monoïde des pomsets.

Diagnostic
ENTRÉES :
- un modèle α, exprimé à l'aide d'une expression rationnelle de $\mathbb{P}(\Sigma, D)$;
- un ensemble d'actions observables Σ_o ;
- une observation o, exprimée à l'aide d'un LPO $(E_o, \leq_o, \lambda_o)$, telle que $\lambda_o(E_o) \subseteq \Sigma_o$.

SORTIE : Quels sont tous les pomsets p tels que $\pi_{\Sigma_o}(p) \in [\![o]\!]_{\Sigma_o,\alpha}$] ?

Le second problème consiste à inférer toutes les causalités non observées. Ici, comme le nombre d'événements observés est borné, le nombre de causalités que l'on va pouvoir rajouter est, lui-aussi, borné. Nous nous contenterons, donc, de donner cette solution sous la forme d'une extension d'ordre de l'observation : toutes les causalités rajoutées doivent être expliquées par toutes les exécutions du modèle, cohérentes avec l'observation. L'objet retourné est, par conséquent, complet vis-à-vis du modèle et de l'observation : on en a tiré toutes les informations de causalité possibles.

Corrélation d'événements
ENTRÉES :
- un modèle α, exprimé à l'aide d'une expression rationnelle de $\mathbb{P}(\Sigma, D)$;
- un ensemble d'actions observables Σ_o ;
- une observation o, exprimée à l'aide d'un LPO $(E_o, \leq_o, \lambda_o)$, telle que $\lambda_o(E_o) \subseteq \Sigma_o$.

SORTIE : Quelle est la plus grande relation $\leq$ telle que $[\![(E_o, \leq, \lambda_o)]\!]_{\Sigma_o,\alpha} = [\![o]\!]_{\Sigma_o,\alpha}$?

6.2 Diagnostic

Comme énoncé précédemment, l'objectif principal du diagnostic est de mettre en place une technique, qui, à partir d'un modèle et d'une observation, construit un générateur de toutes les explications possibles compatibles avec ce modèle et cette observation. Au lieu de répondre explicitement à la question "Quels sont tous les pomsets p tels que $\pi_{\Sigma_o}(p) \in [[\![o]\!]_{\Sigma_o,\alpha}]$?", nous construisons directement un rationnel de pomsets $\alpha_{\Sigma_o,o}$ tel que :

$$\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o,o}) = \{p \mid \pi_{\Sigma_o}(p) \in [[\![o]\!]_{\Sigma_o,\alpha}]\}$$

6.2.1 Diagnostic centralisé

Dans un premier temps, pour répondre à la question du diagnostic dans un cadre centralisé, nous utilisons le modèle des *boxed pomsets* présenté au chapitre 4. Rappelons que ce modèle est tout à fait adapté pour travailler avec des projections de langages de pomsets : en effet, nous avons montré que la projection d'un langage rationnel de boxed pomsets restait un langage rationnel de boxed pomsets, et qu'il était possible de caractériser la projection d'un langage rationnel de pomsets à l'aide d'un langage rationnel de boxed pomsets et d'opérateurs d'encapsulation et de désencapsulation. En effet, rappelons le théorème principal du chapitre 4, à savoir le théorème 4.10, avec $B : \mathbb{P}(\Sigma, D) \to \mathbb{B}(\Sigma, D)$ l'opération d'encapsulation qui permet de passer des pomsets aux boxed pomsets, et $U : \mathbb{B}(\Sigma, D) \to \mathbb{P}(\Sigma, D)$ l'opération inverse :

Soient α, un rationnel de $\mathbb{P}(\Sigma, D)$ et $\Sigma_o \subseteq \Sigma$. Alors :

$$\pi_{\Sigma_o}(\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha)) = U(\mathcal{L}_{\mathbb{B}(\Sigma,D)}(\widehat{\pi}_{\Sigma_o}(B(\alpha))))$$

Nous disposons, ainsi, d'une expression rationnelle de $\mathbb{B}(\Sigma, D)$, $\beta = \widehat{\pi}_{\Sigma_o}(B(\alpha))$, qui permet de générer, à une opération de désencapsulation près, la projection sur Σ_o, de toutes les exécutions de α. Il suffit alors de déplier cette structure, en ne conservant que les exécutions qui expliquent o, pour obtenir le générateur voulu. Ainsi, malgré le fait qu'une exécution puisse contenir un nombre, a priori, non borné d'événements, le dépliage de β restreint à o est fini. Il suffira ensuite de regarder le dépliage restreint équivalent pour α, pour pouvoir générer toutes les exécutions de α expliquant o.

Mais, tout d'abord, nous montrons que le problème du diagnostic est difficile.

Théorème 6.3 *Soient Σ, un ensemble d'actions, $\Sigma_o \subseteq \Sigma$, un ensemble d'actions observables, $D \subseteq \Sigma^2$, une relation, α, une expression rationnelle de $\mathbb{P}(\Sigma, D)$, et o, un* LPO *$(E_o, \leq_o, \lambda_o)$ tel*

que $\lambda_o(E_o) \subseteq \Sigma_o$.

Si α *et* o *ne sont pas auto-concurrents, alors savoir si* o *possède une* α*-explication est NP-complet.*

Preuve: Le problème est NP-complet pour les HMSC, lorsque que $\Sigma_o = \Sigma$. Le problème que nous considérons ici est donc au moins NP.

Montrons, d'une part, que l'on peut vérifier une solution, donnée en terme d'un pomset dont la projection est une α-explication de o, en un temps polynomial en la taille des entrées. Pour cela, notons que, comme α et o ne sont pas auto-concurrents, il est possible de vérifier en temps linéaire qu'un élément de $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha)$ est isomorphe à o. Il suffit, en effet, de les comparer inductivement, en considérant leurs éléments minimaux qui auront des étiquettes différentes.

Ensuite, nous montrons que, pour toute solution, il en existe une équivalente, de taille polynomiale en la taille des entrées, qu'il suffira de vérifier à la place de la première.

Tout d'abord, notons $\mathcal{A}_\alpha$ l'automate associé à α. Une *solution* est un chemin ρ de $\mathcal{A}_\alpha$, étiqueté par u et tel que $\pi_{\Sigma_o}(\mathcal{L}_{\mathbb{P}(\Sigma,D)}(u)) = [E, \leq, \lambda]$ soit une classe d'isomorphisme qui contienne une explication de o.

Considérons, ensuite, $\beta = \widehat{\pi}_{\Sigma_o}(B(\alpha))$ et $\mathcal{A}_\beta$ l'automate structurellement équivalent à β. Le théorème 4.10 nous indique que $\pi_{\Sigma_o}(\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha)) = U(\mathcal{L}_{\mathbb{B}(\Sigma,D)}(\beta))$. Ainsi, ρ est aussi un chemin de $\mathcal{A}_\beta$ étiqueté par v, tel que $U(\mathcal{L}_{\mathbb{B}(\Sigma,D)}(v)) = [E, \leq]$ soit une classe d'isomorphisme qui contienne une explication de o.

Supposons, maintenant que la taille de ρ est plus grande que $|o||\alpha||\Sigma|^2$. Comme ρ a au plus $|o|$ transitions contenant des événements observables, nous pouvons trouver une séquence de transitions ne contenant que des événement inobservables de taille plus grande que $|\alpha||\Sigma|^2$. Il existe donc un cycle dans β, étiqueté par un boxed pomset b, dont la boîte interne est vide, qui apparaît plus de $|\Sigma|$ fois sur cette séquence de transitions inobservables.

Finalement, pour tous boxed pomsets b_i et b_j dont la boîte interne est vide, nous avons : $b_i^{|\Sigma|+1} = b_i^{|\Sigma|}$ et $b_i \boxdot_D b_j = b_j \boxdot_D b_i$, ce qui signifie que nous pouvons enlever des instances du cycle étiqueté par b, pour qu'il apparaisse moins de $|\Sigma|$ fois sur cette séquence. Nous pouvons donc construire un chemin ρ', borné par $|o||\alpha||\Sigma|^2$, qui soit solution. Ceci conclut la preuve. □

Ensuite, nous donnons une méthode pour construire effectivement le diagnostic. Comme énoncé précédemment, cette méthode se base sur la notion de dépliage guidé. Intuitivement, un dépliage guidé est un dépliage d'automate classique, dans lequel on s'autorise à effacer certaines branches liées à des choix. Pour cela, nous définissons, de manière classique, la notion de préfixe pour les boxed pomsets : soit b un boxed pomset b_1 est un *préfixe* de b si il existe b_2 tel que $b = b_1 \boxdot_D b_2$.

Définition 6.4 (dépliage guidé) *Soient* α *et* β*, deux expressions rationnelles de boxed pomsets,* $\mathcal{A}_\alpha = (S, \rightarrow, \mathcal{B}, S_i, S_f)$ *et* $\mathcal{A}_\beta = (S', \Rightarrow, \mathcal{B}, S'_i, S'_f)$*, leurs automates structurellement équivalents et* o *un boxed* LPO*.*

Alors, β *est le* dépliage guidé *par* o *de* α *si :*

– *chaque état de* $\mathcal{A}_\beta$ *est associé à un état de* $\mathcal{A}_\alpha$ *et à un préfixe de* o *:* $S' \subseteq S \times \mathbb{B}(\Sigma, D)$ *;*

- *les états initiaux de $\mathcal{A}_\beta$ sont les états initiaux de $\mathcal{A}_\alpha$ associés au boxed* Lpo *vide, c'est-à-dire : $S'_i = S_i \times \{\varepsilon\}$;*
- *les états finaux de $\mathcal{A}_\beta$ sont les états finaux de $\mathcal{A}_\alpha$ associés à o : $S'_f = S_f \times \{o\}$;*
- *la transition $(s,m) \overset{a}{\Rightarrow} (t,n)$ est dans $\mathcal{A}_\beta$ si, et seulement si, la transition $s \xrightarrow{a} t$ existe dans $\mathcal{A}_\alpha$ et si n peut être expliqué par la concaténation de m et de a, c'est-à-dire si $U([m] \boxdot_D a) \in [\![U(n)]\!]$.*

Dans la suite, nous notons $\mathcal{D}(\alpha, o)$ le dépliage de α guidé par o.

Il est important de remarquer que, par construction, pour tout α et o, nous avons :

$$U(\mathcal{L}_{\mathbb{B}(\Sigma,D)}(\mathcal{D}(\alpha,o))) = U(\mathcal{L}_{\mathbb{B}(\Sigma,D)}(\alpha)) \cap [\![U(o)]\!]$$

c'est-à-dire que la désencapsulation du langage du dépliage guidé $\mathcal{D}(\alpha, o)$ est exactement l'intersection entre la désencapsulation du langage de α et les explications de o.

Ensuite, la figure 6.5 donne un exemple de dépliage guidé par $B(o)$ (avec o, l'observation de la figure 6.3(a)) d'un modèle $\delta = b_1^\star b_2^+$, avec $\Sigma = \{a, b, c\}$ et $D = \{(a,a), (c,b), (c,c)\}$. Les états de l'automate déplié contiennent un état de l'automate $\mathcal{A}_\delta$ ainsi qu'un préfixe P de $B(o)$, codé par la donnée, pour chaque événement du port de sortie P, des événements maximaux de o qu'il domine. Par exemple, l'étiquette "s_1 $(\{2\}, \{1,5\}, \{5\})$", d'un état, qui apparaît dans le dépliage guidé de la figure 6.5, indique que le préfixe P lié a cet état est un boxed pomset ayant un port de sortie :

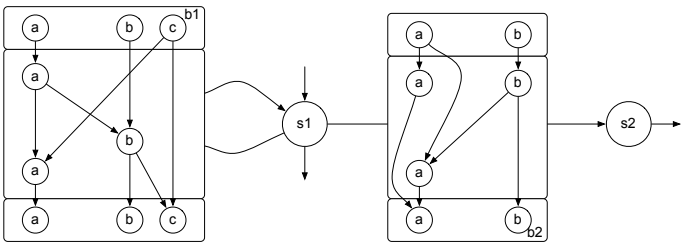
- dont l'événement étiqueté par a domine l'événement 2 (et donc aussi l'événement 1) de l'observation de la figure 6.3(a) ;
- dont l'événement étiqueté par b domine les événements 1 et 5 ;
- dont l'événement étiqueté par a domine l'événement 5.

Considérons maintenant, un modèle α décrit sous la forme d'une expression rationnelle de pomsets et une observation o. Nous pouvons alors construire le dépliage, guidé par $B(o)$, de la structure $\widehat{\pi}_{\Sigma_o}(B(\alpha))$, qui représente, de manière finie, la projection de α sur Σ_o. Puis, nous pouvons, à partir de cette structure, retrouver les pomsets qui ont été projetés pour construire un dépliage, guidé par $B(o)$, de α, c'est-à-dire :

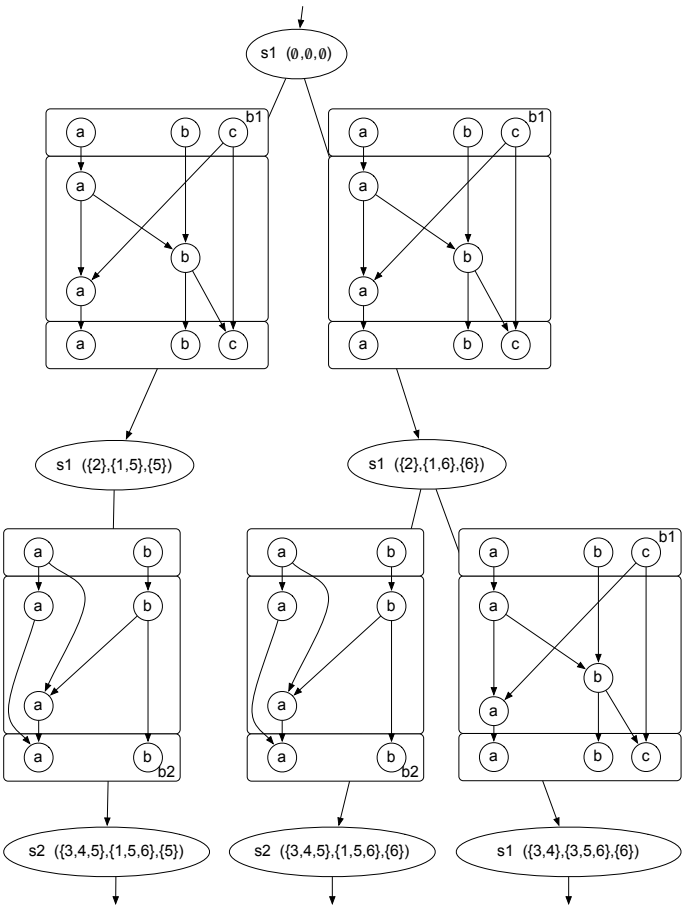
$$\left(\widehat{\pi}_{\Sigma_o} B\right)^{-1} \left(\mathcal{D}\left(\widehat{\pi}_{\Sigma_o}\left(B\left(\alpha\right)\right), B(o)\right)\right)$$

Notons que, dans ce cas bien particulier, calculer $\left(\widehat{\pi}_{\Sigma_o} B\right)^{-1}$ est facile : en effet, il suffit simplement d'annoter, lors de l'encapsulation et de la projection, le nom des pomsets projetés. Notons aussi que, dans le cas des pomsets, certains cycles "silencieux" (car ne produisant pas d'événements observables) peuvent être conservés, et générer un nombre non borné d'événements avant projection. Nous montrons, maintenant, que la structure définie ci-dessus est exactement $\alpha_{\Sigma_o,o}$, le générateur fini que nous cherchons à construire.

Avant de montrer cela formellement, intéressons nous à un exemple. Nous pouvons remarquer que le dépliage construit à la figure 6.5 correspond à $\mathcal{D}(\widehat{\pi}_{\Sigma_o}(B(\alpha)), B(o))$, avec α, o et Σ_o définis dans la figure 6.4. L'expression rationnelle correspondant à l'automate décrit dans cette figure est $b_1 b_1 + b_1(b_1 + b_2)$, ce qui correspond, après application de $\left(\widehat{\pi}_{\Sigma_o} B\right)^{-1}$, à l'expression rationnelle $p_1 p_1 + p_1(p_1 + p_2)$. Le langage de ce rationnel est exactement $\{p \mid \pi(p) \in [\![o]\!]_{\Sigma_o,\alpha}]\}$, c'est-à-dire $\alpha_{\Sigma_o,o}$. Le théorème suivant montre que cet exemple peut se généraliser.



(a) l'automate $\mathcal{A}_\delta$, pour $\delta = b_1^\star b_2^+$, avec $\Sigma = \{a, b, c\}$ et $D = \{(a, a), (c, b), (c, c)\}$.



(b) l'automate correspondant à $\mathcal{D}(\delta, B(o))$.

FIG. *6.5 – Dépliage guidé par $B(o)$ (de la figure 6.3(a)) du modèle δ.*

Théorème 6.5 *Soient Σ, un ensemble d'actions, $\Sigma_o \subseteq \Sigma$, un ensemble des actions observables, $D \subseteq \Sigma^2$, une relation, α, une expression rationnelle de $\mathbb{P}(\Sigma, D)$, appelée modèle et o, un* LPO *$(E_o, \leq_o, \lambda_o)$ tel que $\lambda_o(E_o) \subseteq \Sigma_o$, appelé observation. Alors :*

$$\alpha_{\Sigma_o,o} = (\widehat{\pi}_{\Sigma_o} B)^{-1} \left(\mathcal{D}\left(\widehat{\pi}_{\Sigma_o}\Big(B\left(\alpha\right)\Big), B(o) \right) \right)$$

Preuve: Dans cette preuve, notons $\beta = (\widehat{\pi}_{\Sigma_o} B)^{-1} \left(\mathcal{D} \left(\widehat{\pi}_{\Sigma_o} \left(B \left(\alpha \right) \right), B(o) \right) \right)$.

D'une part, montrons l'inclusion $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o,o}) \subseteq \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\beta)$. Pour cela, prenons un p dans $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o,o})$, c'est-à-dire tel que $\pi_{\Sigma_o}(p) \in [\![o]\!]_{\Sigma_o,\alpha}]$. Par définition, $\pi_{\Sigma_o}(p) \in [\![o]\!]_{\Sigma_o,\alpha}]$ si, et seulement si :

$$\pi_{\Sigma_o}(p) \in [\![o]\!]] \text{ et } \pi_{\Sigma_o}(p) \in \pi_{\Sigma_o}(\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha))$$

Ce qui implique, en utilisant les propositions 4.4 et 4.5 et le théorème 4.10 :

$$\pi_{\Sigma_o}(p) \in [\![U(B(o))]\!]] \text{ et } \pi_{\Sigma_o}(p) \in U(\mathcal{L}_{\mathbb{B}(\Sigma,D)}(\widehat{\pi}_{\Sigma_o}(B(\alpha))))$$

Donc, $\pi_{\Sigma_o}(p) \in U(\mathcal{D}(\widehat{\pi}_{\Sigma_o}(B(\alpha)), B(o)))$. On peut ensuite montrer que $\pi_{\Sigma_o}^{-1} U = (\pi_{\Sigma_o} U B)^{-1} U = (\widehat{\pi}_{\Sigma_o} B)^{-1}$, et donc que $p \in \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\beta)$.

Inversement, montrons que $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\beta) \subseteq \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o,o})$. Ceci est immédiat car le dépliage guidé ne génère que des sous-expressions de α. □

Proposition 6.6 *Soient Σ, un ensemble d'actions, $\Sigma_o \subseteq \Sigma$, un ensemble d'actions observables, $D \subseteq \Sigma^2$, une relation, α, une expression rationnelle de $\mathbb{P}(\Sigma, D)$, et o, un* Lpo *$(E_o, \leq_o, \lambda_o)$ tel que $\lambda_o(E_o) \subseteq \Sigma_o$.*

Si o n'est pas auto-concurrent et que Σ est fixé, alors le problème du diagnostic est dans NLOGSPACE. Plus précisément, on peut construire une structure de taille $O(|\alpha| \cdot |o|^{|\Sigma||\Sigma_o|})$ qui génère toutes les exécutions dont les projections sont des α-explications de o.

Preuve: Il suffit de considérer la structure construite dans le théorème 6.5, dont la taille est en $O(|\alpha| \cdot |o|^{|\Sigma||\Sigma_o|})$ si o n'est pas auto-concurrent : en effet, pour se rappeler un préfixe de $B(o)$, il suffit pour chaque $\sigma \in \Sigma$, que le port de sortie du préfixe, étiqueté par σ, retienne les lettres observables qu'il domine. □

Les théorèmes précédents nous montrent que la complexité du diagnostic dépend, avec un facteur exponentiel, de la taille de l'alphabet observé. Cependant, il existe des cas où aucune explication ne permet d'exprimer une observation et dans ces cas, il n'est pas forcément nécessaire de construire de si grosses structures pour le découvrir. En effet, il est possible, dans le cadre réparti, de diviser l'alphabet d'observation centralisé en plusieurs alphabet répartis et plus petits, et de résoudre le problème du diagnostic pour chacun de ces alphabets d'observation.

6.2.2 Diagnostic réparti

Nous nous allons intéressons, maintenant, au diagnostic réparti. Comme expliqué dans la partie 6.1.1, nous découpons, maintenant, l'alphabet global d'observation en plusieurs alphabets disjoints, et nous allons attacher chacun à un système de log différent. Chacun de ces systèmes de log va pouvoir réaliser l'algorithme centralisé, présenté dans la partie précédente, mais uniquement sur les événements qu'il observe. La difficulté consiste, ensuite, à fusionner les structures obtenues, pour obtenir un diagnostic global complet. La solution à ce problème

est, en fait, toute simple : il suffit de faire le produit des dépliages obtenus.

Plus formellement, soient $\Sigma_o^i \subseteq \Sigma_o$ un ensemble d'actions locales observables, et $o = (E_o, \leq_o, \lambda_o)$ une observation. Le diagnostic local pour l'ensemble d'actions Σ_o^i est l'expression rationnelle $\alpha_{\Sigma^i,o}$ telle que $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o^i,o}) = [\![\pi_{\Sigma_o^i}(o)]\!]_{\Sigma_o^i,\alpha}$. Comme une explication pour un alphabet Σ est toujours une explication pour un alphabet $\Sigma' \subseteq \Sigma$, nous avons $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o,o}) \subseteq \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o^i,o})$. On obtient donc :

$$\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o^i \cup \Sigma_o^j,o}) \subseteq \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o^i,o}) \cap \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o^j,o})$$

Ceci signifie qu'un diagnostic plus fin peut être obtenu en combinant les diagnostics locaux. Dans le cadre du diagnostic, nous montrons que cette combinaison est simplement le produit $\times$ des automates associés aux expressions rationnelles manipulées. Nous montrons aussi que cette combinaison devient exacte lorsque les alphabets d'observation locaux sont choisis correctement.

Plus précisément, étant donné deux diagnostics locaux $\alpha_{\Sigma_o^i}$ et $\alpha_{\Sigma_o^j}$, notons $\mathcal{A}_i$ et $\mathcal{A}_j$ les automates structurellement identiques correspondants. Le diagnostic produit, $\alpha_{\Sigma_o^i} \times \alpha_{\Sigma_o^j}$, sera alors l'expression rationnelle correspondant à l'automate $\mathcal{A}_k = \mathcal{A}_i \times \mathcal{A}_j$, défini comme suit :

- l'ensemble des états, états initiaux et états finaux de $\mathcal{A}_k$ est le produit cartésien des états, états initiaux et états finaux de $\mathcal{A}_i$ et $\mathcal{A}_j$;
- la transition $(s_i, s_j) \rightarrow (s_i', s_j')$ est dans $\mathcal{A}_k$ si, et seulement si, la transition $s_i \rightarrow s_i'$ est dans $\mathcal{A}_i$ et la transition $s_j \rightarrow s_j'$ est dans $\mathcal{A}_j$.

Remarquons que $\times$ est associative et commutative, ce qui assure que nous pouvons effectuer la reconstruction globale dans n'importe quel ordre. Notons aussi, que s'il existe un sous-ensemble Σ_o^i tel que $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o^i,o}) = \emptyset$, alors la reconstruction globale $\alpha_{\Sigma_o,o}$ aura, elle aussi, un langage vide (c.a.d ne pourra pas expliquer l'observation). Plus important, la proposition suivante indique que si une exécution appartient à tous les langages des $\alpha_{\Sigma_o^i,o}$, alors elle appartient, aussi, au langage de $\alpha_{\Sigma_o,o}$.

Proposition 6.7 *Soient Σ, un ensemble d'actions, $\Sigma_o^i \subseteq \Sigma$, des ensembles d'actions observables pour $1 \leq i \leq n$, $\Sigma_o = \bigcup_{1 \leq i \leq n} \Sigma_o^i$, $D \subseteq \Sigma^2$, une relation, α, une expression rationnelle de $\mathbb{P}(\Sigma, D)$, et o, un* Lpo *$(E_o, \leq_o, \lambda_o)$ tel que $\lambda_o(E_o) \subseteq \Sigma_o$.*

Si, pour tout couple $(\sigma, \sigma') \in \Sigma_o \times \Sigma_o$, il existe un i tel que $\{\sigma, \sigma'\} \subseteq \Sigma_o^i$, alors :

$$\alpha_{\Sigma_o,o} = \alpha_{\Sigma_o^1,o} \times \ldots \times \alpha_{\Sigma_o^n,o}$$

Preuve: D'une part, nous avons $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o,o}) \subseteq \bigcap_{1 \leq i \leq n} \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o^i,o})$, et donc $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o,o}) \subseteq \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o^1,o} \times \ldots \times \alpha_{\Sigma_o^n,o})$.

D'autre part, supposons qu'il existe un p dans $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o^1,o} \times \ldots \times \alpha_{\Sigma_o^n,o})$ qui ne soit pas dans $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o,o})$. Cela signifie que p est dans $\bigcap_{1 \leq i \leq n} \mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o^i,o})$, et donc que p est dans $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o^i,o})$, pour tout i. Si p n'est pas dans $\mathcal{L}_{\mathbb{P}(\Sigma,D)}(\alpha_{\Sigma_o,o})$, cela veut alors dire qu'il existe e, e' dans E_p tel que $e \leq_p e'$ mais que $\pi_{\Sigma_o}(p)$ ne soit pas une explication de o. Or, il existe un i tel que $\{\lambda_p(e), \lambda(e')\} \subseteq \Sigma_o^i$, avec $\pi_{\Sigma_o^i}(p)$ une explication de o. Contradiction. □

Pour conclure sur ces aspects répartis, remarquons que pour faire baisser la complexité locale au minimum, tout en conservant un diagnostic complet, il suffit de choisir l'alphabet réparti $\Sigma_o^{ij} = \{\sigma_i, \sigma_j\}$ (avec $\Sigma = \{\sigma_1 \dots \sigma_{|\Sigma|}\}$. Ce découpage permet de produire plus rapidement un résultat négatif, quand il n'existe pas d'explications possibles à une observation. Mais dans le cas contraire, la complexité est la même que dans le cadre centralisé, même si la complexité moyenne devrait être meilleure en pratique.

6.3 Corrélation d'événements

Nous avons vu, dans la partie précédente, comment construire un générateur fini de toutes les exécutions dont la projection sur l'alphabet observable explique l'observation. Ce générateur nous permet donc d'inférer l'ensemble des événements non observés. Dans cette partie, nous allons reprendre les mêmes raisonnements, mais en nous concentrant sur l'inférence de causalités non observées. Plus précisément, nous allons chercher à ajouter des causalités sur l'observation dont nous sommes sûrs qu'elles sont valides, c'est-à-dire si ces causalités apparaissent dans toutes les explications de l'observation.

Nous pouvons donc reformuler le problème de la corrélation d'événements de la manière suivante :

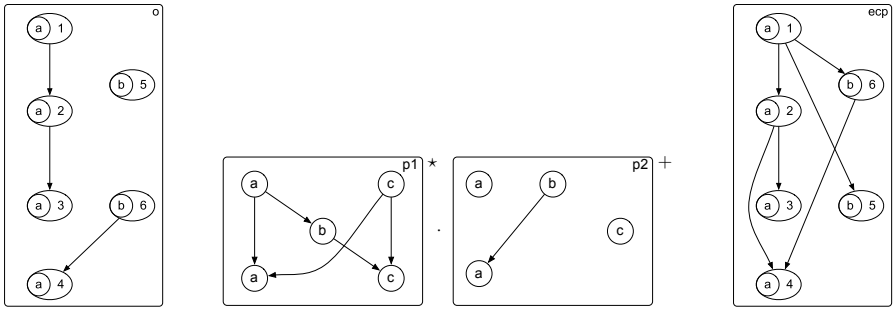
Définition 6.8 (PCE) *Soient Σ, un ensemble d'actions, $D \subseteq \Sigma^2$, α, une expression rationnelle de $\mathbb{P}(\Sigma, D)$, $\Sigma_o \subseteq \Sigma$, un ensemble d'actions observables, $o = (E_o, \leq_o)$, un* Lpo*, et (e_1, e_2), une paire d'événements de E_o.*

Le Problème de la Corrélation d'Evénements *pour (e_1, e_2), noté $PCE(\Sigma_o, \mathcal{A}, o, e_1, e_2)$, est de décider si $e_1 \leq e_2$ pour tout $p = (E_o, \leq) \in [\![o]\!]_{\Sigma_o, \mathcal{A}}$. Nous noterons $pce_{\Sigma_o,\alpha,o}$ le* Lpo *$(E_o, \leq)$, où $e_1 \leq e_2$ si, et seulement si, $PCE(\Sigma_o, \alpha, o, e_1, e_2)$.*

L'ensemble des *causes* d'une observation est l'ensemble des événements minimaux de $pce_{\Sigma_o,\alpha,o}$. Rajouter des causalités dans l'observation permet donc de diminuer les causes du phénomène observé. C'est l'argument principal qui motive la suite de ce chapitre.

Nous considérons maintenant le problème de la corrélation d'événements dans un cadre centralisé. Cette technique consiste à inférer des causalités lors de l'observation partielle de l'exécution d'un système par un unique système de log. Rappelons que l'objectif d'une telle analyse est de diminuer le nombre d'événements minimaux d'une observation, et donc de limiter les causes possibles d'une séquence d'alarmes observées. Pour faire cela, nous considérons que nous disposons d'un modèle, donné sous la forme d'une expression rationnelle de pomsets et nous allons utiliser la construction présentée dans la partie précédente pour ajouter les causalités inférées.

La figure 6.6 donne un exemple de corrélation d'événements, avec le modèle α et l'observation o déjà utilisée dans la figure 6.4. Les causalités ajoutées entre les paires d'événements $(1, 5)$, $(1, 6)$ et $(2, 4)$ apparaissent dans toutes les α-explications de o (voir la figure 6.4(c)). De plus, dans ce cas, la cause de o est l'événement 1, étiqueté par a.



(a) observation (b) modèle avec $D = \{(a,a),(b,c),(c,c)\}$ (c) $pce_{\{a,b\},\alpha,o}$

FIG. *6.6 – Observations, modèle et corrélations*

Nous allons maintenant montrer, dans le théorème 6.9, que PCE est un problème difficile. Cependant, nous montrerons ensuite, avec le théorème 6.11, qu'il existe un cas dans lequel PCE peut être résolu en temps polynomial. De plus, la preuve est constructive, ce qui conduit directement à un algorithme effectif.

Théorème 6.9 *Soient Σ, un ensemble d'actions, $D \subseteq \Sigma^2$, α, une expression rationnelle de $\mathbb{P}(\Sigma, D)$, $\Sigma_o \subseteq \Sigma$, un ensemble d'actions observables, $o = (E_o, \leq_o)$, un* Lpo*, et (e, e'), une paire d'événements de E_o.*

Alors, $PCE(\Sigma_o, \alpha, o, e, e')$ est CoNP-complet.

Preuve: Nous voulons montrer que répondre à la question suivante est un problème NP-complet : Existe-t-il un Lpo lpo $l = (E_o, \leq) \in [\![o]\!]_{\Sigma_o,\alpha}$ tel que $e \not\leq e'$?

D'une part, montrons que ce problème est dans NP. Une solution à ce problème est un couple (l, p) tel que $e \not\leq_l e'$ et $\pi_{\Sigma_o}(p) = [l]$. Il faut ensuite montrer que l'on peut choisir un p' équivalent à p mais borné par $|\alpha||\Sigma|^2$. Ceci est fait de manière identique à celle du théorème 6.5. Ce qui montre que l'on peut vérifier une solution en temps polynomial.

D'autre part, montrons que ce problème est au moins NP. Pour cela, il est immédiat de se ramener à un problème d'appartenance pour les hmsc. □

Cependant, nous montrons, qu'en utilisant une structure très proche du dépliage guidé présenté dans la partie précédente, nous pouvons résoudre efficacement ce problème dans le cas où Σ est fixé. Avant d'exposer la définition d'un *moniteur guidé*, notons que les opérations habituelles sur les pomsets ou les boxed pomsets peuvent être étendues directement aux Lpo ou aux boxed Lpo en prenant en considération le nom précis des événements manipulés.

Définition 6.10 (moniteur guidé) *Soient α, une expression rationnelle de boxed pomsets, β, une expressions rationnelle de boxed* Lpo*, $\mathcal{A}_\alpha = (S, \rightarrow, \mathcal{B}, S_i, S_f)$ et $\mathcal{A}_\beta = (S', \Rightarrow, \widehat{\mathcal{B}}, S'_i, S'_f)$, leurs automates structurellement équivalents et o, un boxed* Lpo*.*

Alors, β est le moniteur guidé *par o de α si :*
– *chaque état de $\mathcal{A}_\beta$ est associé à un état de $\mathcal{A}_\alpha$ et à un préfixe de o : $S' \subseteq S \times \mathbb{B}(\Sigma, D)$;*

- *les états initiaux de $\mathcal{A}_\beta$ sont les états initiaux de $\mathcal{A}_\alpha$ associés au boxed* Lpo *vide, c'est-à-dire : $S'_i = S_i \times \{\varepsilon\}$;*
- *les états finaux de $\mathcal{A}_\beta$ sont les états finaux de $\mathcal{A}_\alpha$ associés à o : $S'_f = S_f \times \{o\}$;*
- *la transition $(s,m) \overset{a}{\Rightarrow} (t,n)$ est dans $\mathcal{A}_\beta$ si, et seulement si, la transition $s \xrightarrow{[a]} t$ existe dans $\mathcal{A}_\alpha$ et si n peut être expliqué par la concaténation de m et de a, c'est-à-dire si $U(m \boxdot_D a) \in [\![U(n)]\!]$.*

Dans la suite, nous noterons $\mathcal{M}(\alpha, o)$ le moniteur de α guidé par o.

Le moniteur guidé possède pratiquement les mêmes propriétés que le dépliage guidé défini dans la partie précédente. Cependant, un moniteur guidé génère des Lpo qui ont tous les mêmes événements E_o et qui ne diffèrent que par les causalités qui les relient. Ainsi, il n'est pas très difficile de voir que :

$$\mathcal{L}(\mathcal{M}(\widehat{\pi}_{\Sigma_o}(B(\alpha), B(o)))) = [\![o]\!]_{\Sigma_o,\alpha}$$

La figure 6.7 montre le moniteur guidé par o du modèle δ, qui correspond à la version boxed Lpo du dépliage guidé par o que montre la figure 6.5. Les transitions de cette structure contiennent des Lpo dont les événements sont des événements de o. Il suffit, ensuite, pour obtenir le résultat de la figure 6.6(c), d'intersecter toutes les explications produites par le moniteur.

Le théorème suivant indique que cette construction marche en effet dans tous les cas, et il donne un algorithme efficace, quand Σ est fixé, pour faire de la corrélation d'événements.

Proposition 6.11 *Soient Σ, un ensemble d'actions, $D \subseteq \Sigma^2$, α, une expression rationnelle de $\mathbb{P}(\Sigma, D)$, $\Sigma_o \subseteq \Sigma$, un ensemble d'actions observables, $o = (E_o, \leq_o)$, un* Lpo*, et (e, e'), une paire d'événements de E_o.*

Si o n'est pas auto-concurrent et si Σ est fixé, alors, pour tout (e, e'), $PCE(\Sigma_o, \alpha, o, e, e')$ est NLOGSPACE. Plus précisément, ce problème peut être résolu en $O(|\alpha||o|^{|\Sigma||\Sigma_o|})$.

Preuve: La preuve nécessite de construire $\mathcal{M}(\widehat{\pi}_{\Sigma_o}(B(\alpha)), B(o))$, et de calculer l'intersection des Lpo $(E_o, \leq, \lambda_o)$ générés par cette structure. Pour la complexité, se reporter à la preuve de la proposition 6.6. □

Pour étendre ces résultats au cadre réparti, il faudrait utiliser et appliquer les mêmes techniques que pour le diagnostic, et communiquer entre les systèmes de log répartis, les moniteurs calculés localement, pour en faire le produit. En effet, il n'est, a priori, pas suffisant de s'échanger une solution locale au problème de la corrélation d'événements.

6.4 Conclusions et perspectives

Résumé

Dans ce chapitre, nous avons développé deux techniques nouvelles de supervision de modèles de haut niveau, avec observation partielle. Ces techniques s'appuient, au niveau

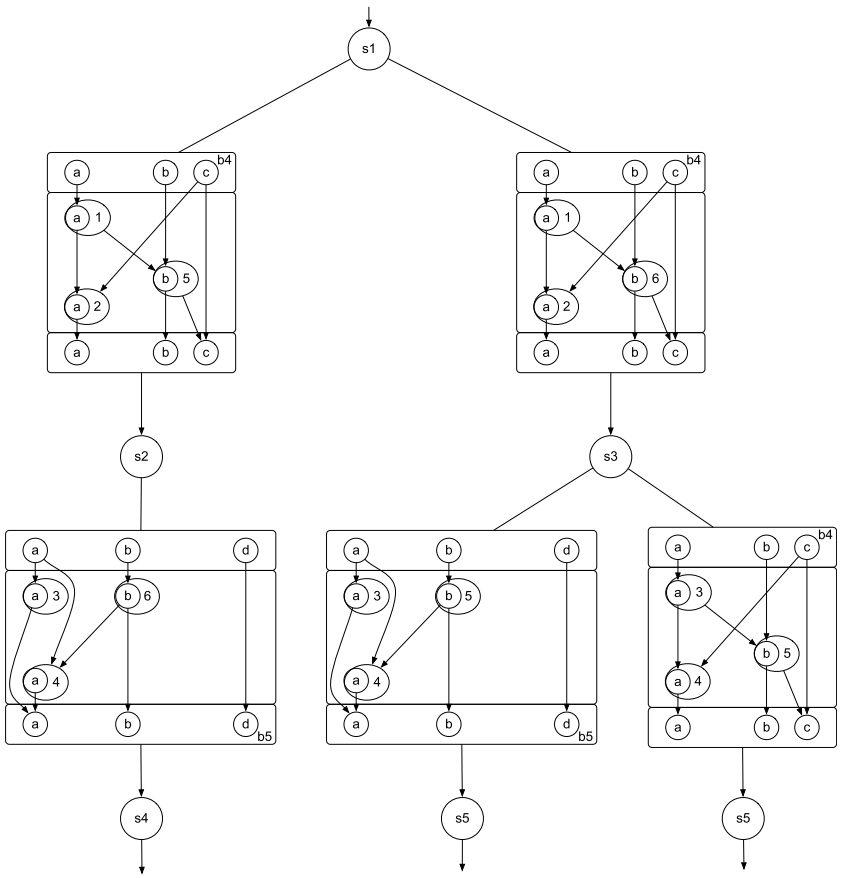


FIG. *6.7 – Moniteur guidé par $B(o)$ du modèle δ (o est défini dans la figure 6.4 et δ dans la figure 6.5).*

théorique, sur le formalisme de projection exacte de modèles développés dans le chapitre 4. Cela est motivé par le fait que nous modélisons les observations partielles par la projection d'une exécution d'un modèle de haut niveau. Les deux techniques que nous avons développées cherchent donc à "inverser" la projection, afin de retrouver, d'une part, les événements non observés et, d'autre part, les causalités non observées.

La première technique que nous avons développée correspond aux techniques classiques de diagnostic fondées sur un modèle, bien que celles-ci ne peuvent s'appliquer dans notre cas puisque nous travaillons sur des modèles de haut niveau quelconques (et donc, a priori, non bornés). De plus, nous exploitons pleinement la sémantique d'ordres partiels de nos modèles, puisqu'à aucun moment nous ne considérons des langages d'entrelacements du modèle. Pour cela, nous avons cherché à reconstruire l'ensemble (infini) des exécutions dont la projection sur les actions observables peut expliquer l'observation, en construisant un générateur fini de ces exécutions. Celui-ci prend la forme d'un dépliage du modèle donné en entrée, guidé par l'observation fournie. Nous avons ensuite montré que, dans le cas où toutes les actions qui ont le même nom sont exécutées par une même entité (et donc par conséquent, que

toutes les occurrences d'une même action sont totalement ordonnées[1]), répondre à la question "existe-t-il une exécution qui explique l'observation ?" est NP-complet. Enfin, nous avons montré que dans le même cas, si Σ est fixé, ce problème est dans NLOGSPACE. Les preuves sont constructives, nous sommes donc capables d'exhiber ce générateur fini. Notons que la condition de non auto-concurrence est en général vérifiée, il suffit de modifier l'étiquette d'un événement enregistré par un capteur en rajoutant le nom de ce capteur. Nous montrons aussi qu'il est possible de répartir le calcul de ce diagnostic tout en restant complet et valide.

La seconde technique que nous avons développée est plus originale, puisqu'elle s'appuie directement sur la représentation par ordre partiel de nos modèles. En effet, nous avons cherché à rajouter, à l'observation, toutes les causalités dont nous sommes sûrs qu'elles sont produites par la projection d'une exécution du modèle sur les événements observables. Nous disposons ainsi d'une représentation finie et facilement compréhensible, permettant d'interpréter l'observation. En particulier, rajouter des causalités signifie diminuer le nombre d'événements minimaux, et donc permettre de réduire les causes probables du comportement observé. Pour résoudre cela, nous utilisons une variante du dépliage guidé, développée dans le cadre du diagnostic, variante que nous avons appelée moniteur guidé. Cette structure nous a permis de montrer que, dans le cas où l'observation n'est pas auto-concurrente, décider si deux événements observés étaient causalement reliés est CoNP-complet, et dans NLOGSPACE si, en plus, la taille de l'alphabet est fixée. Ce résultat est une bonne illustration de la théorie développée dans cette thèse : en effet, il nous semble très difficile d'obtenir le même type de résultat avec des modèles de bas niveau, ou n'ayant pas de sémantique d'ordres partiels.

Applications

Nous avons développé, dans le cadre des techniques de diagnostic développées dans ce chapitre, un prototype appelé `ScenarioDoctor`, écrit en Ocaml (environ 2500 lignes de code) et accessible à l'adresse suivante :

`http://www.irisa.fr/distribcom/Personal_Pages/gazagnaire/tools/`

Ce prototype prend en entrée une description d'un système, sous la forme d'un HMSC causal, un alphabet observable et une observation. En sortie, il produit un HMSC causal dont la projection de toutes les exécutions explique l'observation.

Perspectives

Ces travaux ouvrent deux types de perspectives. Les premières sont spécifiques aux techniques que nous avons développées, les secondes sont plus générales.

Tout d'abord, une perspective spécifique concernant le diagnostic est l'application directe des techniques issues de l'état de l'art sur les automates. En effet, la technique de diagnostic que nous avons développée, dans ce chapitre, montre que les boxed pomsets peuvent être employés avec succès pour traduire les techniques classiques utilisant des projections d'automates vers des techniques utilisant des projections d'expressions rationnelles de pomsets.

[1] Ce que nous avons appelé, dans ce chapitre, "ne pas être auto-concurrent".

L'idée est, donc, de continuer cette traduction et d'essayer de définir ce que pourrait être, dans ce contexte, un diagnostiqueur ([Sampath 96]) à mémoire bornée qui fonctionnerait à la volée. De la même manière, les techniques développées dans [Jéron 06] sembleraient pouvoir s'appliquer aussi, puisqu'elles s'appuient sur des techniques de projection. Cependant, dans ce cadre, la difficulté serait de donner un équivalent aux états marqués employés par les auteurs, car, pour nos modèles, il n'est pas possible de parler d'états globaux.

Ensuite, d'autres perspectives spécifiques concernent la corrélation d'événements.

D'une part, il faudrait regarder, plus en détail, la version répartie de l'algorithme. Transmettre les moniteurs guidés entre les différentes entités, à la manière de la transmission des dépliages guidés présentée pour le diagnostic, permettrait de résoudre le problème de la corrélation répartie d'événements. Cependant, comme nous donnons comme solution une extension d'ordre de l'observation, il existe peut-être des techniques qui nécessitent de transmettre moins d'information entre les entités. C'est, en tout cas, un point intéressant à étudier.

D'autre part, il serait intéressant de considérer des modèles probabilistes, par exemple des chaînes de Markov étiquetées par des pomsets, afin de donner une interprétation plus fine de l'observation. En effet, pour le moment, nous rajoutons des causalités quand nous sommes sûrs qu'elles sont présentes dans la projection de toutes les exécutions du modèle. Notons aussi que nous pouvons, en utilisant la même technique, rendre indépendants tous les événements qui sont indépendants dans la projection de toutes les exécutions du modèle. Cependant, nous aimerions être capables de dire qu'une causalité a, par exemple, 45% de chance d'être présente. Il nous semble envisageable de résoudre ce problème, plus complexe que celui auquel nous nous sommes intéressés dans ce chapitre, en utilisant une version probabilisée (comme les "fuzzy posets" définis dans [Neggers 01]) des modèles de haut niveau que nous manipulons.

Finalement, la perspective générale qui découle de ces travaux est la suivante : pour le moment, nous disposons de systèmes de log qui collectent les différentes alarmes qui ont lieu sur le réseau. Au bout d'un certain temps, ces systèmes lancent une phase d'analyse afin, soit de diagnostiquer ce qu'ils ont observé, soit de trouver les causes du phénomène observé. La prochaine étape de ce processus est de pouvoir faire cette détection à la volée.

Dans le cas où le système observé a un comportement qui est finiment engendré, les outils que nous avons donnés dans la première partie du chapitre 5 pour construire des macro-événements à la volée peuvent nous être utiles. En effet, nous pouvons utiliser ces outils comme un pré-traitement, et réaliser un dépliage guidé (ou un moniteur guidé) au fur et à mesure que les macro-événements arrivent.

Cependant, dans le cas général, il nous manque des outils théoriques pour manipuler la fermeture par préfixe de modèles de haut niveau. Dans le cas des CHMSC, le monoïde produit semble être adapté (voir [Caillaud 00]), mais dans le cas général, le bon modèle reste à inventer.

Conclusion

L'objectif de cette thèse était de montrer que les ordres partiels étiquetés étaient le bon formalisme pour *modéliser*, *vérifier* et *superviser* des systèmes parallèles et répartis.

Partie I

Tout d'abord, nous avons montré, dans la partie I, que ce formalisme permettait de décrire globalement les interactions entre les entités de ces systèmes, que ces interactions se fassent par l'intermédiaire de mémoire partagée ou par l'intermédiaire d'échanges de messages. Cette approche prend le contre-pied de l'approche classique, inspirée des systèmes ouverts, qui consiste à modéliser séparément les entités dans un formalisme inspiré des modèles séquentiels classiques, puis à calculer les interactions qui apparaissent lorsque les différentes entités évoluent ensemble. Nous avons donc choisi de modéliser directement les interactions, au lieu de les calculer. Cette approche globale intègre directement à la phase de modélisation, les outils qui en simplifient l'analyse.

Dans le cas des systèmes bornés, nous avons montré que l'approche globale permettait de décrire les mêmes systèmes que l'approche qui consiste à modéliser indépendamment les différentes entités. En effet, nous avons montré que les CHMSC causaux réguliers sont équivalents aux réseaux bornés d'automates communicants. De même, nous avons montré que les CHMSC causaux réguliers et cohérents, sont équivalents aux réseaux bornés d'automates mixtes. A notre sens, ces résultats sont préliminaires : en effet, ils sont pour le moment restreints aux systèmes bornés, ce qui n'est pas totalement satisfaisant et ne permet pas de caractériser exactement les systèmes modélisables par l'approche globale.

Partie II

Ensuite, dans la partie II, nous avons montré que l'approche globale que nous avons développée permet de vérifier des modèles dont l'espace des états n'est pas borné.

D'une part, nous avons montré, que pour la restriction syntaxique des corationnels de pomsets, il était possible de vérifier si un modèle satisfaisait une propriété exprimée dans une logique temporelle classique mais où les propositions atomiques sont des pomsets. Cette méthode de vérification est efficace puisqu'elle a la même complexité que pour le model-checking de systèmes séquentiels vis-à-vis de formules des logiques temporelles classiques. Comme l'approche globale est plus concise que l'approche bas niveau, ce résultat indique en fait que le model-checking de modèles globaux est exponentiellement plus efficace que le

model-checking de modèles séquentiels. Enfin, cette analyse permet de vérifier, de manière exacte, des modèles dont l'espace des états est infini, ce qui est un résultat très satisfaisant. Ceci constitue le principal résultat théorique de cette thèse.

D'autre part, nous avons montré, que pour certains modèles, il était possible de faire de la vérification partielle. Pour cela, nous utilisons le modèle des *boxed pomsets*, pour lequel la projection est distributive sur la composition. Ce nouveau modèle est un outil très efficace pour montrer des résultats sur des modèles partiels. En effet, nous l'utilisons pour montrer qu'il est possible de faire du model-checking (pour le moment, uniquement positif) de vues partielles de certains modèles, exprimés à l'aide de corationnels de pomsets. Ce modèle est ensuite utilisé dans la partie III, lorsque nous avons fait de la supervision d'observation partielle. Nous pensons que le modèle des boxed pomsets est un outil très utile pour manipuler à la fois des modèles de haut niveau et des vues partielles de ces modèles.

Partie III

Enfin, dans la partie III, nous avons montré que des problèmes de supervision, qui sont difficiles, voire impossibles à réaliser lorsque l'on modélise indépendamment chaque entité, se résolvent simplement dans le cas de ces modèles globaux.

D'une part, nous avons proposé une approche de compression et d'analyse, fondée sur la structure d'ordre partiel des exécutions. Nous avons, ensuite, utilisé cette technique pour proposer une heuristique pour savoir si cette exécution compressée avait pu être générée par un modèle. Cette technique peut être vue comme la recherche de modèles pour lesquels tester l'appartenance d'un vecteur à leur image de Parikh peut être réalisé en temps polynomial. Nous avons donc identifié une classe de modèles globaux, ayant un *indice de Parikh* de 0, pour lesquels nous pouvons décider, en temps polynomial, s'il existe une exécution du modèle qui explique une observation donnée. L'approche que nous proposons est intéressante pour aider à la conception de systèmes dont la supervision sera facile. Cependant, elle est limitée par l'absence de réels outils théoriques pour modéliser le langage des préfixes d'une expression rationnelle de pomsets. En son absence, les solutions algorithmiques que nous donnons et qui s'exécutent en ligne, nécessitent des hypothèses très fortes sur le système sous-jacent (qu'il soit finiment engendré, par exemple), ce qui n'est pas totalement satisfaisant.

D'autre part, nous avons montré, qu'il était possible, à partir d'un modèle et d'une observation partielle, de construire un générateur fini qui engendre exactement l'ensemble (infini) des exécutions de ce modèle qui expliquent l'observation. Ce générateur permet, ensuite, de généraliser cette approche à une construction répartie pour le calcul d'un diagnostic. Ensuite, toujours à partir d'un modèle global et d'une observation partielle, nous avons montré qu'il était possible d'inférer les causalités induites par le modèle et non observées de cette observation partielle. Augmenter la causalité permet de diminuer le nombre d'événements minimaux et donc le nombre de causes probables du phénomène observé. Ces deux approches sont intéressantes d'un point de vue algorithmique : en effet, premièrement, la construction du générateur permet de répartir les calculs de diagnostic sur plusieurs observateurs répartis tout en conservant un résultat d'analyse exact. Deuxièmement, l'approche fondée sur les modèles de haut niveau que nous avons choisie pour résoudre le problème de la corrélation

d'événements, démontre le pouvoir de tels modèles, puisqu'il n'est pas possible de répondre à cette question dans le cadre de systèmes qui ne manipulent pas des ordre partiels. Enfin, d'un point de vue théorique, les résultats démontrés sont un bel exemple de l'utilité des boxed pomsets : ils sont utilisés pour construire le générateur fini utilisé pour le diagnostic et pour la corrélation d'événements.

Perspectives

De nombreuses perspectives ont été données dans les différents chapitres de cette thèse. Nous ne les reprenons pas ici. Nous nous concentrons sur deux perspectives plus générales, liées à la conception de modèles de haut niveau.

En effet, les contributions que nous avons évoquées précédemment mettent en lumière l'intérêt qu'offrent les modèles de haut niveau pour l'analyse des systèmes parallèles et répartis. Analyse qui est aussi bien statique, avec de la vérification, que dynamique, avec de la supervision. Cependant, ne nous leurrons pas. Cette facilité d'analyse indique simplement que nous avons reporté les difficultés vers la phase de modélisation : nous avons complexifié les modèles pour que ceux-ci deviennent plus simples à vérifier et à superviser.

La question qu'il faut maintenant se poser est la suivante :

Est-il facile de modéliser un système parallèle et réparti à l'aide de modèles globaux ?

La réponse à cette question est très simple : *tout dépend des outils de spécification.* De ce point de vue, deux perspectives se dessinent. La première perspective, un peu utopiste, consiste à construire un nouvel environnement de développement entièrement dédié à la conception de systèmes parallèles et répartis. Cet environnement faciliterait la mise au point de modèle de haut niveau et générerait automatiquement les programmes de bas niveau correspondant. La seconde perspective, un peu plus réaliste, consiste à l'inverse, à construire ou à inférer des modèles de haut niveau à partir de programmes déjà existants.

Un langage dédié à la programmation de systèmes parallèles et répartis

Les modèles de haut niveau que nous décrivons dans cette thèse sont très généraux, mais ils ne permettent pas de manipuler directement des données : puisqu'il n'y a pas de notion de variable, les itérations ne correspondent pas à des boucles `for` et les choix ne correspondent pas des `if`. Les comportements que nous modélisons ne sont que des abstractions de comportements possibles de tels systèmes.

Afin de proposer un outil de modélisation proche d'un langage de programmation pour les systèmes parallèles et répartis, il faudrait, donc, enrichir nos modèles avec des constructions classiques pour la prise en compte des données : variables, piles, ... et étudier les modifications à apporter pour que les traductions vers des modèles de bas niveau que nous avons proposées dans le chapitre 2, restent valides. Ces méthodes permettraient de générer directement des programmes à compiler et à exécuter sur les différentes entités, pour obtenir un

scénario d'interaction spécifié par le modèle global.

Dans cet objectif, nous avons commencé à étudier cette piste de recherche en proposant un langage, nommé DISTRIL, qui permet de programmer globalement des comportements entre différentes entités. Pour le moment, le langage possède juste des variables et une notion de procédures. Cependant, celles-ci ne sont pas récursives et peuvent être dépliées statiquement, ce qui nous permet de les enlever. Ainsi, nous avons commencé à mettre en œuvre un "compilateur" de ce langage vers un modèle de haut niveau, décrit à l'aide de rationnels de pomsets et à programmer les algorithmes distribués classiques (élection, consensus).

La phase suivante est de fournir une interface vers un outil de model-checking, afin de mettre en œuvre les différentes techniques présentées dans la partie II, puis d'utiliser les outils développés dans la partie I pour compiler les modèles de haut niveau construits vers des programmes de bas niveau. Enfin, il faudrait intégrer à cette phase de compilation les outils d'observation et de supervision que nous avons développés dans la partie III.

A terme, l'objectif est d'obtenir une plate-forme complète de développement et d'analyse pour les systèmes parallèles et répartis. Nous pensons que les modèles de haut niveau ont atteint une maturité suffisante pour qu'ils puissent être utilisables en pratique.

Construire un modèle de haut niveau à partir de programmes déjà existants

L'approche inverse de la perspective précédente est de se reposer sur les programmes déjà existants et de construire, à partir d'eux, un modèle de haut niveau que l'on va pouvoir analyser. Cette approche est séduisante puisqu'elle ne nécessite pas, pour les analyser, d'écrire d'une nouvelle façon des programmes déjà existants.

Cette approche a été suivie par exemple par Chatain et al. dans [Chatain 05], pour les systèmes bornés. La méthode qu'ils ont suivie, c'est-à-dire l'identification de points de coupure dans une exécution pour reconstruire les atomes d'un HMSC, pourrait être généralisée grâce à des techniques d'abstraction du contenu des canaux de communication par des langages réguliers, développés par le Gall et al. dans [Gall 06].

Cependant, les résultats d'équivalence entre modèles de bas niveau et modèles de haut niveau présentés dans la partie I de ce manuscrit, indiquent qu'une telle traduction automatique, si elle est faisable dans certains cas, est, soit, impossible, soit très coûteuse (avec au moins deux facteurs exponentiels dans le meilleur des cas) dans le cas général.

Il faut donc se contenter de méthodes heuristiques qui vont essayer de construire un modèle le plus exact possible, avec un maximum de concision. De telles approches sont, en général, basées sur l'apprentissage : on laisse les entités interagir et on essaye d'apprendre un modèle de haut niveau en les observant. C'est, par exemple, l'approche qui a été suivie récemment par Jourdan et al. dans [Jourdan 07] et par Bollig et al. qui ont développé un prototype pour faire de l'apprentissage de scénario, nommé SMYLE ([Bollig 07]).

Au final, bien que certains outils de modélisation existent (citons SOFAT ([Hélouët 99]) et MSCAN ([Bollig 06]), qui permettent d'analyser des spécifications données à l'aide de HMSC), il devient réellement nécessaire d'avoir une plate-forme unifiée et complète de modélisation et d'analyse des systèmes parallèles et répartis. Nous espérons que le cadre que nous avons présenté dans ce document pourra aider à la création d'une telle plate-forme.

Bibliographie

[Ahuja 90] Mohan Ahuja, Ajay D. Kshemkalyani, Timothy Carlson. – A Basic Unit of Computation in Distributed Systems. – *ICDCS*, pp. 12–19. IEEE Computer Society, 1990.

[Alur 95] Rajeev Alur, Doron Peled, Wojciech Penczek. – Model-Checking of Causality Properties. – *LICS*, pp. 90–100. IEEE Computer Society, 1995.

[Alur 96] Rajeev Alur, Gerard J. Holzmann, Doron Peled. – An Analyser for Mesage Sequence Charts. – *TACAS*, édité par Tiziana Margaria, Bernhard Steffen, vol. 1055 de *Lecture Notes in Computer Science*, pp. 35–48. Springer, 1996.

[Alur 99] Rajeev Alur, Mihalis Yannakakis. – Model Checking of Message Sequence Charts. – *CONCUR*, édité par Jos C. M. Baeten, Sjouke Mauw, vol. 1664 de *Lecture Notes in Computer Science*, pp. 114–129. Springer, 1999.

[Alur 00] Rajeev Alur, Kousha Etessami, Mihalis Yannakakis. – Inference of message sequence charts. – *ICSE*, pp. 304–313, 2000.

[Alur 01] Rajeev Alur, Kousha Etessami, Mihalis Yannakakis. – Realizability and Verification of MSC Graphs. – *ICALP*, édité par Fernando Orejas, Paul G. Spirakis, Jan van Leeuwen, vol. 2076 de *Lecture Notes in Computer Science*, pp. 797–808. Springer, 2001.

[Alur 03] Rajeev Alur, Swarat Chaudhuri, Kousha Etessami, Sudipto Guha, Mihalis Yannakakis. – Compression of Partially Ordered Strings. – *CONCUR*, édité par Roberto M. Amadio, Denis Lugiez, vol. 2761 de *Lecture Notes in Computer Science*, pp. 42–56. Springer, 2003.

[Basten 97] Twan Basten, Thomas Kunz, James P. Black, Michael H. Coffin, David J. Taylor. – Vector Time and Causality Among Abstract Events in Distributed Computations. *Distributed Computing*, 11(1) :21–39, 1997.

[Baudru 03] Nicolas Baudru, Rémi Morin. – Safe Implementability of Regular Message Sequence Chart Specifications. – *SNPD*, édité par Walter Dosch, Roger Y. Lee, pp. 210–217. ACIS, 2003.

[Baudru 05] Nicolas Baudru, Rémi Morin. – Polynomial Synthesis of Asynchronous Automata. *CoRR*, abs/cs/0506096, 2005.

[Baudru 07] Nicolas Baudru, Rémi Morin. – Synthesis of Safe Message-Passing Systems. – *FSTTCS*, édité par Vikraman Arvind, Sanjiva Prasad, vol. 4855 de *Lecture Notes in Computer Science*, pp. 277–289. Springer, 2007.

[BenAbdallah 97] Hanêne Ben-Abdallah, Stefan Leue. – Syntactic Detection of Process Divergence and Non-local Choice in Message Sequence Charts. – *TACAS*, édité par Ed Brinksma, vol. 1217 de *Lecture Notes in Computer Science*, pp. 259–274. Springer, 1997.

[Benveniste 03] A. Benveniste, E. Fabre, C. Jard, S. Haar. – Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5) :714–727, May 2003.

[Berard 01] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen. – *Systems and software verification : model-checking techniques and tools.* – Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[Blackburn 06] Patrick Blackburn, Johan F. A. K. van Benthem, Frank Wolter. – *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning).* – Elsevier Science Inc., New York, NY, USA, 2006.

[Bollig 05] Benedikt Bollig. – On the Expressiveness of Asynchronous Cellular Automata. – *FCT*, édité par Maciej Liskiewicz, Rüdiger Reischuk, vol. 3623 de *Lecture Notes in Computer Science*, pp. 528–539. Springer, 2005.

[Bollig 06] Benedikt Bollig, Carsten Kern, Markus Schlütter, Volker Stolz. – MSCan - A Tool for Analyzing MSC Specifications. – *TACAS*, édité par Holger Hermanns, Jens Palsberg, vol. 3920 de *Lecture Notes in Computer Science*, pp. 455–458. Springer, 2006.

[Bollig 07] Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker. – Replaying Play In and Play Out : Synthesis of Design Models from Scenarios by Learning. – *TACAS*, édité par Orna Grumberg, Michael Huth, vol. 4424 de *Lecture Notes in Computer Science*, pp. 435–450. Springer, 2007.

[Brand 83] Daniel Brand, Pitro Zafiropulo. – On Communicating Finite-State Machines. *Journal of the ACM*, 30(2) :323–342, 1983.

[Caillaud 99] B. Caillaud. – Bounded Petri-net synthesis techniques and their applications to the distribution of reactive automata. *JESA, European Journal on Automated Systems*, 33(8) :925–942, 1999.

[Caillaud 00] Benoît Caillaud, Philippe Darondeau, Loïc Hélouët, Gilles Lesventes. – HMSCs as Partial Specifications ... with PNs as Completions. – *MOVEP*, édité par Franck Cassez, Claude Jard, Brigitte Rozoy, Mark Dermot Ryan, vol. 2067 de *Lecture Notes in Computer Science*, pp. 125–152. Springer, 2000.

[Cartier 69] P. Cartier, D. Foata. – *Problèmes combinatoires de commutation et réarrangements.* – Springer-Verlag, iv+88p., Berlin, 1969.

[Chatain 05] Thomas Chatain, Loïc Hélouët, Claude Jard. – From Automata Networks to HMSCs : A Reverse Model Engineering Perspective. – *FORTE*, édité par Farn Wang, vol. 3731 de *Lecture Notes in Computer Science*, pp. 489–502. Springer, 2005.

[Clarke 86] Edmund M. Clarke, E. Allen Emerson, A. Prasad Sistla. – Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.*, 8(2) :244–263, 1986.

[Clarke 99] Edmund M. Clarke, Orna Grumberg, Doron A. Peled. – *Model Checking*. – MIT Press, Cambridge, MA, 1999.

[Clarke 00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, Helmut Veith. – Counterexample-Guided Abstraction Refinement. – *CAV*, édité par E. Allen Emerson, A. Prasad Sistla, vol. 1855 de *Lecture Notes in Computer Science*, pp. 154–169. Springer, 2000.

[Clifford 61] A. H. Clifford, G. B. Preston. – *The Algebraic Theory of Semigroups, Volume I.* – American Mathematical Society, Providence, Rhode Island, 1961.

[Commoner 71] F. Commoner, Anatol W. Holt, Shimon Even, Amir Pnueli. – Marked Directed Graphs. *J. Comput. Syst. Sci.*, 5(5) :511–523, 1971.

[Damm 99] Werner Damm, David Harel. – LSCs : Breathing Life into Message Sequence Charts. – *FMOODS*, édité par Paolo Ciancarini, Alessandro Fantechi, Roberto Gorrieri, vol. 139 de *IFIP Conference Proceedings*. Kluwer, 1999.

[Darondeau 07] Philippe Darondeau, Blaise Genest2, Loïc Loïc Hélouët. – *Products of Message Sequence Charts*. – Rapport de Recherche n° RR-6258, INRIA, 2007.

[Desel 95] Jörg Desel, Javier Esparza. – *Free choice Petri nets*. – Cambridge University Press, New York, NY, USA, 1995.

[Diekert 95] V. Diekert, G. Rozenberg édité par . – *Book of Traces*. – World Scientific, Singapore, 1995.

[Diekert 01] Volker Diekert, Paul Gastin. – Local Temporal Logic is Expressively Complete for Cograph Dependence Alphabets. – *LPAR*, édité par Robert Nieuwenhuis, Andrei Voronkov, vol. 2250 de *Lecture Notes in Computer Science*, pp. 55–69. Springer, 2001.

[Dousson 99] Christophe Dousson, Thang Vu Duong. – Discovering Chronicles with Numerical Time Constraints from Alarm Logs for Monitoring Dynamic Systems. – *IJCAI*, édité par Thomas Dean, pp. 620–626. Morgan Kaufmann, 1999.

[Emerson 84] E. Allen Emerson, A. Prasad Sistla. – Deciding Branching Time Logic. – *STOC*, pp. 14–24. ACM, 1984.

[Emerson 85] E. Allen Emerson. – Automata, Tableaux and Temporal Logics (Extended Abstract). – *Logic of Programs*, édité par Rohit Parikh, vol. 193 de *Lecture Notes in Computer Science*, pp. 79–88. Springer, 1985.

[Emerson 87] E. Allen Emerson, Chin-Laung Lei. – Modalities for Model Checking : Branching Time Logic Strikes Back. *Sci. Comput. Program.*, 8(3) :275–306, 1987.

[Emerson 88] E. Allen Emerson, Charanjit S. Jutla. – The Complexity of Tree Automata and Logics of Programs (Extended Abstract). – *FOCS*, pp. 328–337. IEEE, 1988.

[Emerson 90] E. Allen Emerson. – Temporal and Modal Logic. *Handbook of Theoretical Computer Science, Volume B : Formal Models and Sematics (B)*, pp. 995–1072. – Elsevier and MIT Press, 1990.

[Ésik 99] Zoltán Ésik, Satoshi Okawa. – Series and Parallel Operations on Pomsets. – *FSTTCS*, édité par C. Pandu Rangan, Venkatesh Raman, Ramaswamy Ramanujam, vol. 1738 de *Lecture Notes in Computer Science*, pp. 316–328. Springer, 1999.

[Esparza 94] Javier Esparza, Mogens Nielsen. – Decidability Issues for Petri Nets - a survey. *Bulletin of the EATCS*, 52 :244–262, 1994.

[Fanchon 02] Jean Fanchon, Rémi Morin. – Regular Sets of Pomsets with Autoconcurrency. – *CONCUR*, édité par Lubos Brim, Petr Jancar, Mojmír Kretínský, Antonín Kucera, vol. 2421 de *Lecture Notes in Computer Science*, pp. 402–417. Springer, 2002.

[Fidge 91] Colin J. Fidge. – Logical Time in Distributed Computing Systems. *IEEE Computer*, 24(8) :28–33, 1991.

[Fischer 79] Michael J. Fischer, Richard E. Ladner. – Propositional Dynamic Logic of Regular Programs. *J. Comput. Syst. Sci.*, 18(2) :194–211, 1979.

[Francez 86] N. Francez. – *Fairness*. – Springer-Verlag, Berlin, Heidelberg, 1986.

[Gall 06] Tristan Le Gall, Bertrand Jeannet, Thierry Jéron. – Verification of Communication Protocols Using Abstract Interpretation of FIFO Queues. – *AMAST*, édité par Michael Johnson, Varmo Vene, vol. 4019 de *Lecture Notes in Computer Science*, pp. 204–219. Springer, 2006.

[Gastin 02] Paul Gastin, Madhavan Mukund. – An Elementary Expressively Complete Temporal Logic for Mazurkiewicz Traces. – *ICALP*, édité par Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, Ricardo Conejo, vol. 2380 de *Lecture Notes in Computer Science*, pp. 938–949. Springer, 2002.

[Gastin 03] Paul Gastin, Dietrich Kuske. – Satisfiability and Model Checking for MSO-definable Temporal Logics are in PSPACE. – *CONCUR*, édité par Roberto M. Amadio, Denis Lugiez, vol. 2761 de *Lecture Notes in Computer Science*, pp. 218–232. Springer, 2003.

[Gazagnaire 07a] T. Gazagnaire, C. Jard. – Abstraire à la volée les événements d'un système réparti. – *NOTERE*, 2007.

[Gazagnaire 07b] Thomas Gazagnaire, Blaise Genest, Loïc Hélouët, P. S. Thiagarajan, Shaofa Yang. – Causal Message Sequence Charts. – *CONCUR*, édité par Luís Caires, Vasco Thudichum Vasconcelos, vol. 4703 de *Lecture Notes in Computer Science*, pp. 166–180. Springer, 2007.

[Gazagnaire 07c] Thomas Gazagnaire, Loïc Hélouët. – Event Correlation with Boxed Pomsets. – *FORTE*, édité par John Derrick, Jüri Vain, vol. 4574 de *Lecture Notes in Computer Science*, pp. 160–176. Springer, 2007.

[Genest 02a] Blaise Genest, Anca Muscholl. – Pattern Matching and Membership for Hierarchical Message Sequence Charts. – *LATIN*, édité par Sergio Rajsbaum, vol. 2286 de *Lecture Notes in Computer Science*, pp. 326–340. Springer, 2002.

[Genest 02b] Blaise Genest, Anca Muscholl, Helmut Seidl, Marc Zeitoun. – Infinite-State High-Level MSCs : Model-Checking and Realizability. – *ICALP*, édité par Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, Ricardo Conejo, vol. 2380 de *Lecture Notes in Computer Science*, pp. 657–668. Springer, 2002.

[Genest 03] Blaise Genest, Loïc Hélouët, Anca Muscholl. – High-Level Message Sequence Charts and Projections. – *CONCUR*, édité par Roberto M. Amadio, Denis Lugiez, vol. 2761 de *Lecture Notes in Computer Science*, pp. 308–322. Springer, 2003.

[Genest 04a] Blaise Genest, Marius Minea, Anca Muscholl, Doron Peled. – Specifying and Verifying Partial Order Properties Using Template MSCs. – *FoSSaCS*, édité par Igor Walukiewicz, vol. 2987 de *Lecture Notes in Computer Science*, pp. 195–210. Springer, 2004.

[Genest 04b] Blaise Genest, Anca Muscholl, Dietrich Kuske. – A Kleene Theorem for a Class of Communicating Automata with Effective Algorithms. – *Developments in Language Theory*, édité par Cristian Calude, Elena Calude, Michael J. Dinneen, vol. 3340 de *Lecture Notes in Computer Science*, pp. 30–48. Springer, 2004.

[Genest 05] Blaise Genest. – *The odyssey of MSC-graphs*. – PhD. Thesis, LIAFA, University Paris 7, 2005.

[Genest 06a] Blaise Genest, Dietrich Kuske, Anca Muscholl. – A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Information and Computation*, 204(6) :920–956, 2006.

[Genest 06b] Blaise Genest, Anca Muscholl. – Constructing Exponential-Size Deterministic Zielonka Automata. – *ICALP (2)*, édité par Michele Bugliesi, Bart Preneel, Vladimiro Sassone, Ingo Wegener, vol. 4052 de *Lecture Notes in Computer Science*, pp. 565–576. Springer, 2006.

[Gischer 88] Jay L. Gischer. – The Equational Theory of Pomsets. *Theoretical Computer Science*, 61 :199–224, 1988.

[Goel 03] Ankit Goel, Abhik Roychoudhury, Tulika Mitra. – Compactly representing parallel program executions. – *PPOPP*, pp. 191–202. ACM, 2003.

[Govindaraju 00] Shankar G. Govindaraju, David L. Dill. – Counterexample-Guided Choice of Projections in Approximate Symbolic Model Checking. – *ICCAD*, édité par Ellen Sentovich, pp. 115–119. IEEE, 2000.

[Group 90] IETF Network Working Group. – *A Simple Network Management Protocol (SNMP)*. – Rapport de recherche, IETF, 1990.

[Gunter 01] Elsa L. Gunter, Anca Muscholl, Doron Peled. – Compositional Message Sequence Charts. – *TACAS*, édité par Tiziana Margaria, Wang Yi, vol. 2031 de *Lecture Notes in Computer Science*, pp. 496–511. Springer, 2001.

[Hélouët 99] Loïc Hélouët. – A simulation model for message sequence charts. – *SDL Forum*, pp. 473–488, 1999.

[Hélouët 00a] L. Hélouët, C. Jard. – Conditions for synthesis of communicating automata from HMSCs. – *FMICS*, Berlin, April 2000. GMD FOKUS.

[Hélouët 00b] Loïc Hélouët, Pierre Le Maigat. – Decomposition of Message Sequence Charts. – *SAM*, édité par Edel Sherratt, pp. 47–60. VERIMAG, IRISA, SDL Forum, 2000.

[Hélouët 06] Loïc Hélouët, Thomas Gazagnaire, Blaise Genest. – Diagnosis from Scenarios. – *WODES*, 2006.

[Hennessy 85] Matthew Hennessy, Robin Milner. – Algebraic Laws for Nondeterminism and Concurrency. *J. ACM*, 32(1) :137–161, 1985.

[Henriksen 05] J.G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni, P.S. Thiagarajan. – A Theory of Regular MSC Languages. *Information and Computation*, 202(1) :1–38, 2005.

[Holzmann 97] Gerard J. Holzmann. – The Model Checker SPIN. *IEEE Trans. Software Eng.*, 23(5) :279–295, 1997.

[Howell 88] Rodney R. Howell, Louis E. Rosier. – Completeness Results for Conflict-Free Vector Replacement Systems. *J. Comput. Syst. Sci.*, 37(3) :349–366, 1988.

[ITUTS 99] ITU-TS. – *ITU-TS Recommendation Z.120 : Message Sequence Chart (MSC)*. – ITU-TS, 1999.

[Jones 75] Neil D. Jones. – Space-Bounded Reducibility among Combinatorial Problems. *J. Comput. Syst. Sci.*, 11(1) :68–85, 1975.

[Jourdan 07] Guy-Vincent Jourdan, Hasan Ural, Shen Wang, Hüsnü Yenigün. – Recovering Repetitive Sub-functions from Observations. – *FORTE*, édité par John Derrick, Jüri Vain, vol. 4574 de *Lecture Notes in Computer Science*, pp. 35–49. Springer, 2007.

[Jéron 06] T. Jéron, H. Marchand, S. Pinchinat, M.O. Cordier. – Supervision Patterns in Discrete Event Systems Diagnosis. – *WODES*, 2006.

[Klarlund 94] Nils Klarlund, Madhavan Mukund, Milind A. Sohoni. – Determinizing Asynchronous Automata. – *ICALP*, édité par Serge Abiteboul, Eli Shamir, vol. 820 de *Lecture Notes in Computer Science*, pp. 130–141. Springer, 1994.

[Kleene 56] S. C. Kleene. – Representation of Events in Nerve Nets and Finite Automata. *Automata Studies*, édité par C. E. Shannon, J. McCarthy, pp. 3–41. – Princeton, New Jersey, Princeton University Press, 1956.

[Kunz 93] Thomas Kunz. – Issues in Event Abstraction. – *PARLE*, édité par Arndt Bode, Mike Reeve, Gottfried Wolf, vol. 694 de *Lecture Notes in Computer Science*, pp. 668–671. Springer, 1993.

[Kupferman 00] Orna Kupferman, Moshe Y. Vardi, Pierre Wolper. – An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2) :312–360, 2000.

[Kurshan 94] Robert P. Kurshan. – *Computer-aided verification of coordinating processes : the automata-theoretic approach.* – Princeton University Press, Princeton, NJ, USA, 1994.

[Kuske 03] D. Kuske. – Regular sets of infinite message sequence charts. *Information and Computation*, 187(1) :80–109, 2003.

[Lamport 78] Leslie Lamport. – Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7) :558–565, 1978.

[Lamport 86] Leslie Lamport. – On Interprocess Communication. *Distributed Computing*, 1(2) :77–101, 1986.

[Lichtenstein 85a] Orna Lichtenstein, Amir Pnueli. – Checking That Finite State Concurrent Programs Satisfy Their Linear Specification. – *POPL*, pp. 97–107, 1985.

[Lichtenstein 85b] Orna Lichtenstein, Amir Pnueli, Lenore D. Zuck. – The Glory of the Past. – *Logic of Programs*, édité par Rohit Parikh, vol. 193 de *Lecture Notes in Computer Science*, pp. 196–218. Springer, 1985.

[Liu 89] Ming T. Liu. – Protocol Engineering. *Advances in Computers*, 29 :79–195, 1989.

[Lohrey 02] Markus Lohrey. – Safe Realizability of High-Level Message Sequence Charts. – *CONCUR*, édité par Lubos Brim, Petr Jancar, Mojmír Kretínský, Antonín Kucera, vol. 2421 de *Lecture Notes in Computer Science*, pp. 177–192. Springer, 2002.

[Mateescu 04] Alexandru Mateescu. – Algebraic Aspects of Parikh Matrices. – *Theory Is Forever*, édité par Juhani Karhumäki, Hermann A. Maurer, Gheorghe Paun, Grzegorz Rozenberg, vol. 3113 de *Lecture Notes in Computer Science*, pp. 170–180. Springer, 2004.

[Mattern 89] Friedemann Mattern. – Virtual Time and Global States of Distributed Systems. – *Workshop on Parallel and Distributed Algorithms*, édité par Cosnard M. et al., pp. 215–226, North-Holland / Elsevier, 1989. – (Reprinted in : Z. Yang, T.A. Marsland (Eds.), "Global States and Time in Distributed Systems", IEEE, 1994, pp. 123-133.).

[Mauw 93] S. Mauw, M. van Wijk, T. Winter. – A Formal Semantics of Synchronous Interworkings, 1993.

[Mauw 94] Sjouke Mauw, Michel A. Reniers. – An Algebraic Semantics of Basic Message Sequence Charts. *Comput. J.*, 37(4) :269–278, 1994.

[Mauw 97] Sjouke Mauw, Michel A. Reniers. – High-level message sequence charts. – *SDL Forum*, édité par Ana R. Cavalli, Amardeo Sarma, pp. 291–306. Elsevier, 1997.

[Mayr 81] Ernst W. Mayr. – Persistence of Vector Replacement Systems is Decidable. *Acta Inf.*, 15 :309–318, 1981.

[Mazurkiewicz 77] A. Mazurkiewicz. – *Concurrent Program Schemes and Their Interpretations.* – Technical Report n° DAIMA PB–78, Aarhus University, 1977.

[McCulloch 43] W.S. McCulloch, W. Pitts. – A logical calculus of ideas immanent in neural activity. *Bulletin of Mathematical Biophysics*, 5 :115–133, 1943.

[Morin 02] Rémi Morin. – Recognizable Sets of Message Sequence Charts. – *STACS*, édité par Helmut Alt, Afonso Ferreira, vol. 2285 de *Lecture Notes in Computer Science*, pp. 523–534. Springer, 2002.

[Mukund 00] Madhavan Mukund, K. Narayan Kumar, Milind A. Sohoni. – Synthesizing Distributed Finite-State Systems from MSCs. – *CONCUR*, édité par Catuscia Palamidessi, vol. 1877 de *Lecture Notes in Computer Science*, pp. 521–535. Springer, 2000.

[Muller 88] David E. Muller, Ahmed Saoudi, Paul E. Schupp. – Weak Alternating Automata Give a Simple Explanation of Why Most Temporal and Dynamic Logics are Decidable in Exponential Time. – *LICS*, pp. 422–427. IEEE Computer Society, 1988.

[Muscholl 98] Anca Muscholl, Doron Peled, Zhendong Su. – Deciding Properties for Message Sequence Charts. – *FoSSaCS*, édité par Maurice Nivat, vol. 1378 de *Lecture Notes in Computer Science*, pp. 226–242. Springer, 1998.

[Muscholl 99] Anca Muscholl, Doron Peled. – Message Sequence Graphs and Decision Problems on Mazurkiewicz Traces. – *MFCS*, édité par Miroslaw Kutylowski, Leszek Pacholski, Tomasz Wierzbicki, vol. 1672 de *Lecture Notes in Computer Science*, pp. 81–91, London, UK, 1999. Springer-Verlag.

[Neggers 01] J. Neggers, Hee Sik Kim. – Fuzzy posets on sets. *Fuzzy Sets and Systems*, 117(3) :391–402, 2001.

[Nicola 90] Rocco De Nicola, Frits W. Vaandrager. – Action versus State based Logics for Transition Systems. – *Semantics of Systems of Concurrent Processes*, édité par Irène Guessarian, vol. 469 de *Lecture Notes in Computer Science*, pp. 407–419. Springer, 1990.

[Nygate 95] Y. A. Nygate. – Event correlation using rule and object based techniques. – *Integrated Network Management*, édité par Adarshpal S. Sethi, Yves Raynaud, Fabienne Faure-Vincent, vol. 11 de *IFIP Conference Proceedings*, pp. 278–289. Chapman & Hall, 1995.

[Ochmanski 85] Edward Ochmanski. – Regular behaviour of concurrent systems. *Bulletin of the EATCS*, 27 :56–67, 1985.

[OMG 03] OMG. – Unified Modeling Language Specification, 2003.

[Owicki 82] Susan S. Owicki, Leslie Lamport. – Proving Liveness Properties of Concurrent Programs. *ACM Trans. Program. Lang. Syst.*, 4(3) :455–495, 1982.

[Papadimitriou 94] Christos M. Papadimitriou. – *Computational complexity.* – Addison-Wesley, Reading, Massachusetts, 1994.

[Parikh 66] Rohit Parikh. – On Context-Free Languages. *J. ACM*, 13(4) :570–581, 1966.

[Peled 00] Doron Peled. – Specification and Verification of Message Sequence Charts. – *FORTE*, édité par Tommaso Bolognesi, Diego Latella, vol. 183 de *IFIP Conference Proceedings*, pp. 139–154. Kluwer, 2000.

[Perrin 95] Dominique Perrin. – Les débuts de la théorie des automates. *Technique et Science Informatique*, 14 :409–433, 1995.

[Petri 62] C. A. Petri. – Fundamentals of a Theory of Asynchronous Information Flow. – *IFIP Congress*, pp. 386–390, 1962.

[Pnueli 77] Amir Pnueli. – The Temporal Logic of Programs. – *FOCS*, pp. 46–57. IEEE, 1977.

[Pnueli 81] Amir Pnueli. – The Temporal Semantics of Concurrent Programs. *Theoretical Computer Science*, 13 :45–60, 1981.

[Pnueli 00] Amir Pnueli. – Keynote Address : Abstraction, Composition, Symmetry, and a Little Deduction : The Remedies to State Explosion. – *CAV*, édité par E. Allen Emerson, A. Prasad Sistla, vol. 1855 de *Lecture Notes in Computer Science*, p. 1. Springer, 2000.

[Pratt 86] V. Pratt. – Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1) :33–71, 1986.

[Reniers 98] Michel Reniers. – *Message Sequence Chart. Syntax and Semantics*. – PhD. Thesis, Eindhoven Univ. of Technology, 1998.

[Reutenauer 90] Christophe Reutenauer. – *The mathematics of Petri nets*. – Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[Rozenberg 97] Grzegorz Rozenberg, Arto Salomaa édité par . – *Handbook of formal languages, vol. 1 : word, language, grammar*. – Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[Sakarovitch 92] Jacques Sakarovitch. – The "Last" Decision Problem for Rational Trace Languages. – *LATIN*, édité par Imre Simon, vol. 583 de *Lecture Notes in Computer Science*, pp. 460–473. Springer, 1992.

[Sampath 96] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, D.C Teneketzis. – Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2) :105–124, 1996.

[Schnoebelen 02] Ph. Schnoebelen. – The Complexity of Temporal Logic Model Checking. – *Advances in Modal Logic*, édité par Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, Michael Zakharyaschev, pp. 393–436. King's College Publications, 2002.

[Schützenberger 55] Marcel-Paul Schützenberger. – Une théorie algébrique du codage. – Séminaire Dubreil. Algèbre et théorie des nombres, 9 (1955-1956), Exposé No. 15, 24 p., 1955.

[Sengupta 02] Bikram Sengupta, Rance Cleaveland. – Triggered Message Sequence Charts. – *SIGSOFT FSE*, pp. 167–176, 2002.

[Shannon 48] C. Shannon. – A mathematical theory of communication. *Bell System Technical Journal*, 27 :379–423, 623–656, 1948.

[Sistla 85] A. Prasad Sistla, Edmund M. Clarke. – The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3) :733–749, 1985.

[Sistla 87] A. Prasad Sistla, Moshe Y. Vardi, Pierre Wolper. – The Complementation Problem for Büchi Automata with Appplications to Temporal Logic. *Theoretical Computer Science*, 49 :217–237, 1987.

[Streett 84] Robert S. Streett, E. Allen Emerson. – The Propositional Mu-Calculus is Elementary. – *ICALP*, édité par Jan Paredaens, vol. 172 de *Lecture Notes in Computer Science*, pp. 465–472. Springer, 1984.

[Tarjan 72] Robert Endre Tarjan. – Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1(2) :146–160, 1972.

[Thiagarajan 94] P. S. Thiagarajan. – A Trace Based Extension of Linear Time Temporal Logic. – *LICS*, pp. 438–447. IEEE Computer Society, 1994.

[Thiagarajan 02] P. S. Thiagarajan, Igor Walukiewicz. – An Expressively Complete Linear Time Temporal Logic for Mazurkiewicz Traces. *Inf. Comput.*, 179(2) :230–249, 2002.

[Thompson 68] Ken Thompson. – Regular Expression Search Algorithm. *Commun. ACM*, 11(6) :419–422, 1968.

[Turner 93] Kenneth J. Turner. – *Using Formal Description Techniques : An Introduction to Estelle, Lotos, and SDL.* – John Wiley & Sons, Inc., New York, NY, USA, 1993.

[vanderAalst 06] Wil M. P. van der Aalst. – Matching observed behavior and modeled behavior : An approach based on Petri nets and integer programming. *Decision Support Systems*, 42(3) :1843–1859, 2006.

[Vardi 86a] Moshe Y. Vardi, Pierre Wolper. – An Automata-Theoretic Approach to Automatic Program Verification (Preliminary Report). – *LICS*, pp. 332–344. IEEE Computer Society, 1986.

[Vardi 86b] Moshe Y. Vardi, Pierre Wolper. – Automata-Theoretic Techniques for Modal Logics of Programs. *J. Comput. Syst. Sci.*, 32(2) :183–221, 1986.

[Vardi 94a] Moshe Y. Vardi. – Nontraditional Applications of Automata Theory. – *TACS*, édité par Masami Hagiya, John C. Mitchell, vol. 789 de *Lecture Notes in Computer Science*, pp. 575–597. Springer, 1994.

[Vardi 94b] Moshe Y. Vardi, Pierre Wolper. – Reasoning About Infinite Computations. *Inf. Comput.*, 115(1) :1–37, 1994.

[vonNeumann 51] J. von Neumann. – The general and logical theory of automata. – *The Hixon Symposium (Sept. 1948, Pasadena).*, édité par editor. L.A. Jeffress, 1951.

[Zielonka 87] Wieslaw Zielonka. – Notes on Finite Asynchronous Automata. *Informatique Théorique et Applications*, 21(2) :99–135, 1987.

[Zielonka 89] Wieslaw Zielonka. – Safe Executions of Recognizable Trace Languages by Asynchronous Automata. – *Logic at Botik*, édité par Albert R. Meyer, Michael A. Taitslin, vol. 363 de *Lecture Notes in Computer Science*, pp. 278–289. Springer, 1989.

Index

Résumé

Cette thèse se place dans le cadre de la modélisation et de l'analyse de systèmes parallèles et répartis. Plus précisément, nous nous intéressons à la modélisation, la vérification et la supervision de systèmes, composés d'entités indépendantes interagissant localement par mémoire partagée et globalement par échange asynchrone de messages. Dans ce contexte, plutôt que de modéliser séparément chaque entité, puis d'analyser les comportements qui peuvent se produire lorsque ces entités interagissent, nous fournissons une théorie permettant de modéliser globalement le système considéré tout en conservant des propriétés de vérification et de supervision décidables. Cette théorie se base sur le formalisme des ordres partiels étiquetés (appelés "pomsets").

Dans ce but, nous définissons le modèle des HMSC causaux qui étend le formalisme des HMSC en autorisant, comme pour les traces de Mazurkiewicz, certains événements à commuter sur chaque processus. Nous montrons, tout d'abord, qu'une restriction syntaxique des HMSC causaux a le même pouvoir d'expression que les réseaux bornés d'automates mixtes, un modèle qui étend les réseaux d'automates asynchrones de Zielonka et les réseaux d'automates communicants. De plus, nous montrons que les méthodes classiques de model-checking de systèmes séquentiels peuvent s'appliquer aux modèles plus concis fondés sur les pomsets, comme les HMSC causaux, sans perte d'efficacité. Enfin, nous proposons des méthodes de traitement efficace d'observations volumineuses d'exécutions réparties, ainsi que des techniques de supervision, telles que le diagnostic ou la corrélation d'événements, qui utilisent des modèles fondés sur les pomsets.

Abstract

Our work is dedicated to the modeling and the analysis of concurrent and distributed systems. More precisely, we seek to model, verify and supervise systems composed by autonomous entities which interact locally through shared memory and globally through asynchronous message passing. In this context, we argue that, instead of modeling each entity independently and then analyzing what happens when all of them interact, it is better to have a complete theory to globally describe a concurrent system, in which verification and supervision analysis are decidable. This theory is based on partially ordered multisets (ie. "pomsets") which is a "true-concurrency" model.

To this end, we defined the causal HMSC model, based on pomsets and which extends the standardized HMSC formalism with constructs from Mazurkiewicz trace theory. We first show that a syntactic restriction of causal HMSC is equivalent to bounded mixed automata, a formalism which extends Zielonka asynchronous automata and communicating finite state machines. We also show that classical model-checking techniques can be extended to pomsets-based formalisms without losing efficacy. This is an important result, as pomsets-based formalisms, such as causal HMSC, are more concise than sequential system formalisms. Finally, we propose efficient methods to analyse large recorded files from distributed executions and we propose supervision techniques such as diagnosis and event correlation, using pomsets-based models of distributed and concurrent systems.